Third homework of TOML

# Introduction

The third project consists in the calibration of an air pollution sersor in a network. Precisely, this sensor measure these data:

* The amount of ozone in Kohm;
* The temperature in Celsius;
* The percentage of relative humidity;
* The amount of nitrogen dioxide;
* The amount of nitrogen monoxide;
* The amount of sulfur dioxide;
* The amount of PM10.

Moreover we have the measurements relative to the O3 concentration in ugr/m^3, this is the Reference station, the value we have to predict for the calibration.

To perform our purpose, we used different machine learning models in order to see which model fits better.

# What we used

To perform this project, we used the followings Python libraries:

* Pandas: a library for data manipulation;
* Scikit learn: a library containing many machine learning models in order to perform our project.

# Data Analysis

Before calibrating the sensor, we analyse the data to verify if there is correlation between the reference station value and the measured ones.

## Reading the data

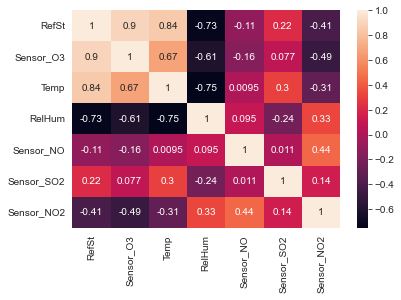
To read the data, we implemented the functions sensorData() and prepareData(), they allow us to read each csv files, make a unique dataframe and add some columns in order to simplify the plotting.

Immagine che contiene testo

Descrizione generata automaticamente Immagine che contiene testo

Descrizione generata automaticamente

## The correlation matrix

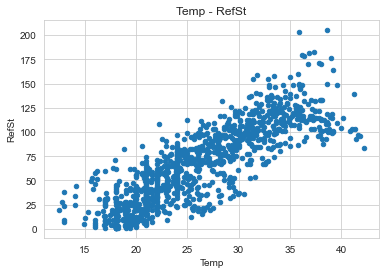
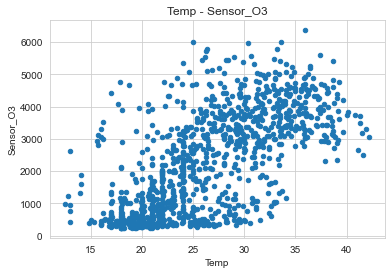


What we can conclude from the correlation matrix above? We can observe that the best features with the highest correlation are Sensor\_O3, Temp, ReHum and Sensor\_NO2, so these features are the one which resume better the value of the reference station. This information will be useful after in this report when we’ll do the feature selection.

But to avoid fast conclusions, let’s go to analyze the plots.

## Plots

### The temperature plots

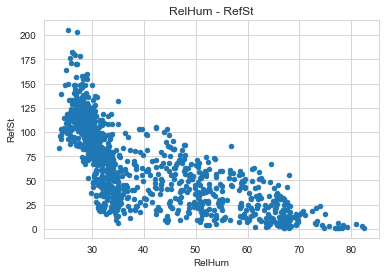
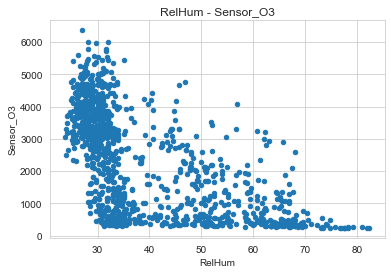
 

The two images on the tops are the plots about the temperature and the value of the O3/Reference station.

As we can see, the shapes in the plots is a little similar, but in the first one is more defined. Always in the first plot, is clearly visible that there is a linear evolution of the data, a thing also visible in the O3 plot but with some noise.

So, we can conclude that the temperature is one of the features that allow to compute the RefSt value in the linear regression.

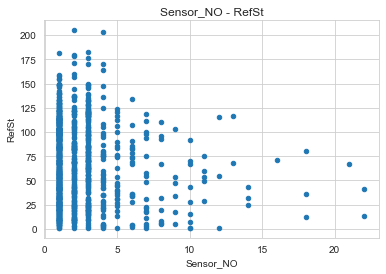
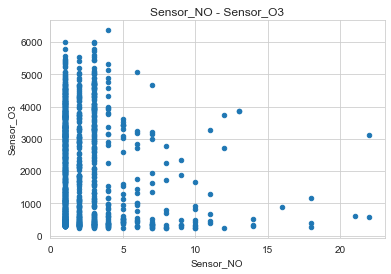
### Humidity plots

Also in this case, we can see that the plots above have more or less the same shape, with more sparsity in the second one. We remember that the humidity is the best negative correlation for RefSt in the negative matrix, this is visible in the plot because the data doesn’t follow any evolution and are concentrated in a region of the plot (in this case when the umidity is around 30). On the other hand, these is another data concentration in the second plot: it is in the low part, where the value of O3 is less than 1000. So, the humidity can be resumed in a hyperbole in the last plot.

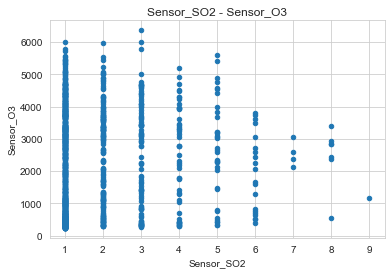
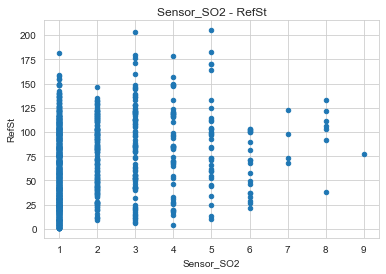
To make some conclusions, Surely the humidity doesn’t impact significantly in the linear regression, but it could inpact in the anothers. We’ll see it later.

### Nitrogen monoxide plots

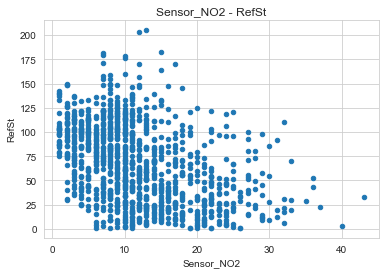
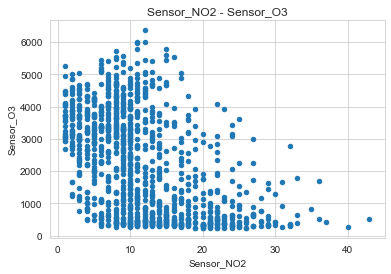
In these plots the data are concentrated in the first values. So, we can make more or less the same conclusions: the NO will impact less in a linear regression and probably it impact more or less at the same way with other algorithms because the correlation is very near to zero respect to the others.

### Sulfur dioxide plots



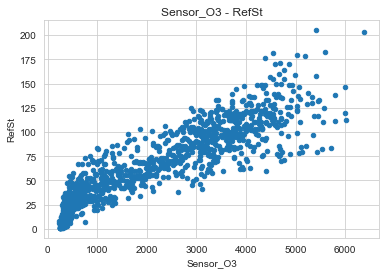
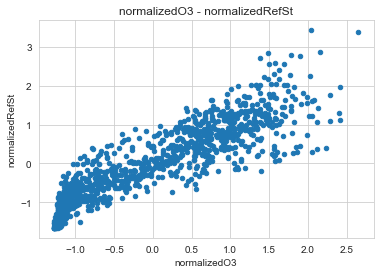
In this situation, we can make the same make the same conclusions of the previous case: because of this feature as a low correlation with RefSt and the plots has not a defined shape, The SO2 is not a good estimator of RefSt.

### Nitrogen Dioxide plots

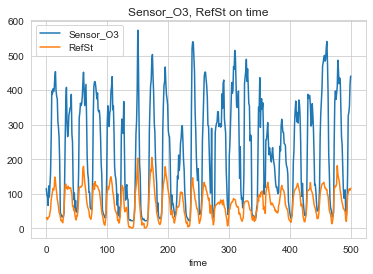
 

This is one of the features we chose from the correlation matrix, we can confirm it because the two plots above present similar shapes. The evolution is not linear, so this feature could be relevant in other algorithms.

### O3 plots

Plotting the O3 values respect to the reference station allows us to see the linear evolution of the last feature. This is a characteristic of an high correlation and we can confirm it with a value of 0.9 on the correlation matrix. As we can see in the second plot, the data normalization



Another confirmation is the plot above, in fact it is clearly visible how the O3 and the reference station follow the same evolution.

NOTE: In order to see this similarity more clearly, the O3 data has been scaled of a factor 50, but it does not affect the evolution! Also we considered only half of the data, but the conclusions we did count for the whole set.

In order to make conclusions, Sensor\_O3 is the best feature to summarize the value of the reference station!

So, for summarizing everything:

* Sensor\_O3 and Temp are the best features!
* Sensor\_NO2 and RelHum are good features!
* Sensor\_NO and Sensor SO2 are bad features.

In the phase of calibration, we considered for each model a 70% of data for the training set and the 30% for the test set.

# Models

For this project we considered the following machine learning algorithms:

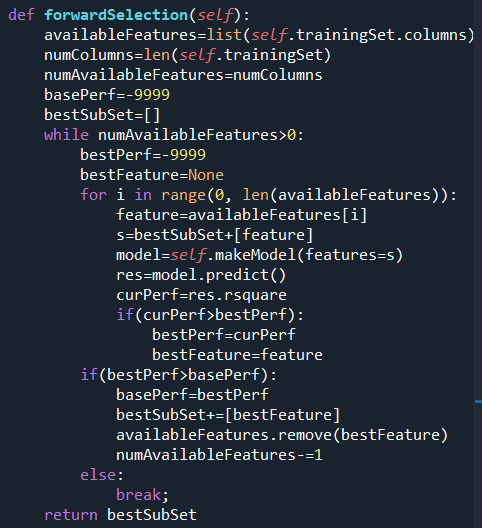
* Normal Linear regression with subset selection:
* Lasso Linear Regression;
* Ridge Linear Regression;
* Kernel Regression;
* KNN Regression;
* Random Forest Regression;
* Support Vector Regression.

The implementation in Python consists in two general classes:

* The class Algorithm is an abstract class where we implemented many general methods that are useful for the training and the predictions;
* The class Model is an abstract class that train the model and make the predictions.

The class Algorithm presents the following methods:

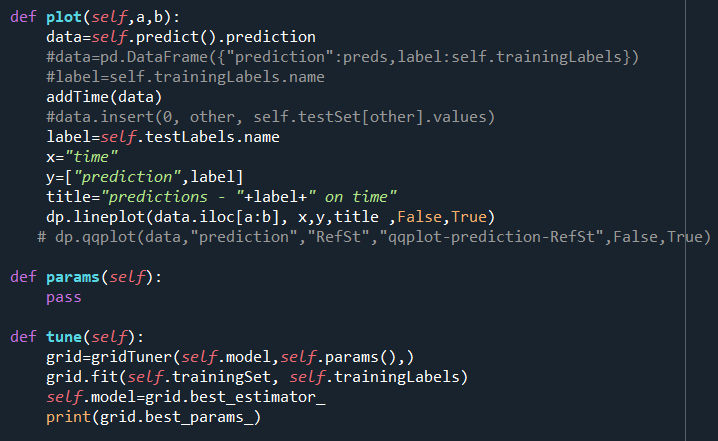
* makeModel(): it creates the model in base of the parameter of the algorithm. Moreover we can specify the features of our interest as parameters. This is an abstract method, so each class extending Algorithm will implement it;
* forwardSelection() analyzes each subset of features and returns the one maximizing the R2 and minimizing the RMSE. This method is useful because it allows us to make the most accurate predictions without falling in the overfitting.



The class Model implements the following methods:

* getCoefficients(): it returns the coefficients of the models;
* getIntercept(): it returns the intercept, the coefficient accrossing the axe y;
* redefineSet(): it adapts the training set and the test set on the features of our interest;
* plot(): make a lineplot in order to compare the predictions with the reference station values;
* tune(): it do the tuning of the hyperparameters in order to increase the accuracy of the model;
* predict(): it makes the predictions basing on the test set and compute the R2, the RMSE and the MAE.

Immagine che contiene testo

Descrizione generata automaticamente

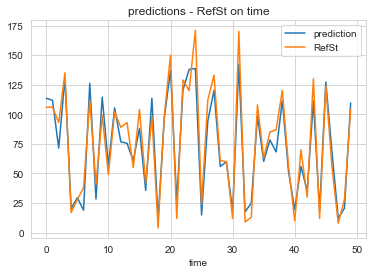
## Linear Regression

The first model we applied is the linear regression but, before doing it, we did a forward subset selection i order to avoid the overfitting. In the data analysis we said that O3 and temperature are the best features of this model, but the forward selection algorithm we implemented returns five features: O3, the temperature, SO2, NO and NO2.

This subset of features allow us to get the best R2 and a reduction of the dataframe by two columns.

## Results

|  |  |  |
| --- | --- | --- |
| R2 | RMSE | MAE |
| 0.917 | 11.636 | 9.026 |



Considering the table and the plot above, we can conclude this is a very nice model! The subset selection allowed us to get more or less the 92% or R2, thing that is clearly visible in the plot because the prediction fits very well with the reference station values.

### Coefficients

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| B0 (Intercept) | B1 (O3) | B2 (Temp) | B3 (SO2) | B4 (NO) | B5 (NO2) |
| -42.306 | 0.175 | 2.738 | 1.06 | -0.45 | 0.178 |

## Lasso Linear Regression

Now we apply always the linear regression but we do a Lasso regularization instead of using the subset selection.

For the definition of Lasso and Ridge regression (we’ll see this one after), we have a parameter alpha working a a weight for the norm of the vector of coefficients.

This is an hyperparameter of the model, so to improve the performance we have to do the tuning.

### Tuning

The tuning is a way to improve the performance of a model changing its hyperparameters. In order to search the best value of alpha, we have tried all the values we can see in the image below.

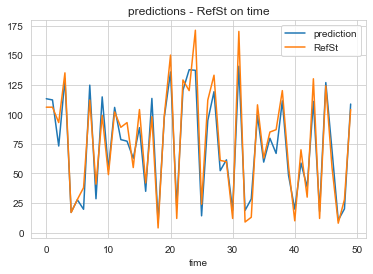
Immagine che contiene testo

Descrizione generata automaticamente

From this set of values, we keep the one giving us the best performances, in case of the Lasso, the best alpha is 0.000001.

### Results

|  |  |  |
| --- | --- | --- |
| R2 | RMSE | MAE |
| 0.917 | 11.66 | 9.044 |



Also in this case we got a a model fitting very well. Comparing these results with the normal linear regression, we can see that this one is less precise but it fits a little better in some parts, in any case these differences are not significative.

### Coefficients

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 |
| -24.27 | 0.17 | 2.426 | -0.232 | -0.384 | 0.953 | 0.192 |

## Ridge Linear Regression

The last linear regression we’ll use is the one which regularize using the Ridge. Also in this case we have an hyperparameter alpha that works exactly at the same way of the Lasso.

### Tuning

We used the same set of values to find the best value of alpha improving the performances of the model. Because of the ridge regression use a two norm, the best value of alpha will be different from the Lasso, in fact it is 100 instead of 0.000001.

### Results

|  |  |  |
| --- | --- | --- |
| R2 | RMSE | MAE |
| 0.917 | 11.67 | 9.045 |

### 

To make some conclusion about the ridge regression, we can say more or less the same things about the Lasso because the results are equivalent! In fact the value of R2, RMSE and MAE differ very little and the plots does not present differences.

So, to conclude this part about the linear regression, we can say that all the models we implemented could be used without problems for the calibration of the sensor, in fact the result are very similar and each one fit almost perfectly the data. Momentaneosly, the linear regression is the best way to calibrate the sensor but let’s go to analyze the behaviour of other models.

### Coefficients

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 |
| -23.58 | 0.172 | 2.41 | -0.237 | -0.372 | 0.89 | 0.19 |

## KNN Regression

Now we go to apply the KNN regression, a model predicting the values finding the best K nearest point to the input. Also in this case we use the forward selection algorithm to choose a subset of feature to use in the prediction. For this algorithm, the best subset is formed by O3, the temperature, SO2 and NO, curiously they are the feature with the highest correlation!

KNN models are very different from the linear ones, in fact we have some hyperparameters:

* the number of neighbors we have to compute;
* the weight function indicate how are weighted the points. KNN can weight the points in a uniform way or using a distance function (so, the nearest points will be more inflent), but we can also use a user-defined function;
* the size of a leaf;
* the power parameter of the Minkowski metric p, in base of the values the model will use a different distance function;

### Tuning

We did the tuning of the hyperparameter using the function below, from this we found the following best values:

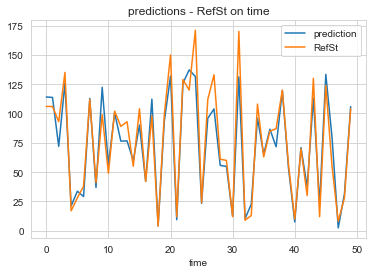
* a leaf size of 10;
* a number of neighbors to compute equals to 10;
* a p equals to 1, so the model will use the Manhattan distance;
* the use of a distance function, so the weights will not be uniform.

Immagine che contiene testo

Descrizione generata automaticamente

### Results

|  |  |  |
| --- | --- | --- |
| R2 | RMSE | MAE |
| 0.9004 | 12.761 | 9.301 |



Thank to the feature selection and the tuning, we got a model fitting very well the data. The accuracy is clearly visible in the plot above, in fact the predictions and the reference station values follows the same evolution with differences in some parts that are not a problem in any case. Comparing these results with the ones we got with the linear models, we can see that this model is less precise, in any case this difference is very little, so it doesn’t matter.

To conclude this part, we can say the linear models remains the best for the sensor calibration, but the KNN one give results very similar, so it is a good alternative to accomplish this purpose.

## Kernel Regression

Now we go to apply the kernel regression, an extension of the linear regression allowing us to regret linearly a non-linear dataset using a function called kernel. Precisely, we use the radial basis function as kernel, it is a function where the values depends only on the distance from the origin or a defined center.

Also in this model we have some hyperparameters, they are the followings:

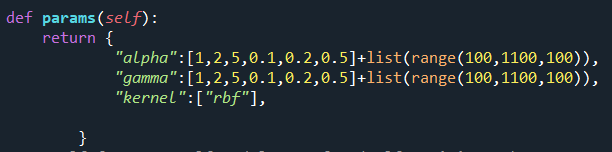
* alpha is the regularization factor;
* the kernel, it is the rbf in our case;
* gamma is the parameter for the rbf.

As for the KNN model, we have to do the feature selection, in this case the temperature results the only feature allowing us to increase the performances.

### Tuning

Now we do the tuning of the hyperparameter, precisely we have to search the best values for alpha and gamma because the kernel is fixed.

In the function below, we can see all the possible vales we tried for each hyperparameter.

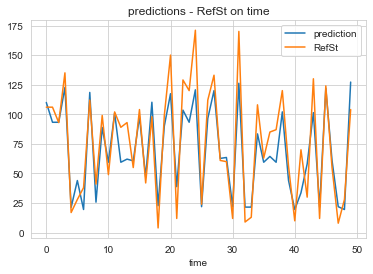


The best value for the hyperparameters are the followings:

* 0.5 for the alpha;
* 0.1 for the gamma.

### Results

|  |  |  |
| --- | --- | --- |
| R2 | RMSE | MAE |
| 0.755 | 20.025 | 15.668 |



Analysing the results we got, we can say this is a good model, in fact we can see from the plot that the prediction does not fit almost perfectly the reference station, but it has the more or less the same evolution.

Comparing this model with the previouses, in this case the differences are notable respect the KNN and the linear model.

So we can conclude that the kernel model can be considered good for the calibration, but in a real case it’s better to try other models in order to find better results. Momentaneosuly this is the worst model for the sensor calibration.

## Random forest regression

Now we try to calibrate the sensor using a random forest, an ensemble method that where we generate a certain number of random trees, each one computes a solution and, from these, we compute the average in case of the regression.

Also for this model we have some hyperparameters, they are very much, so we considered this subset:

* The number of estimators, that is the number of trees in the forest;
* The criterion to measure the quality of the split;
* The max depth of each tree;
* The minimum number of samples to split an internal node;
* The minimum number of samples to split a leaf;
* The number of feature considered for the split.

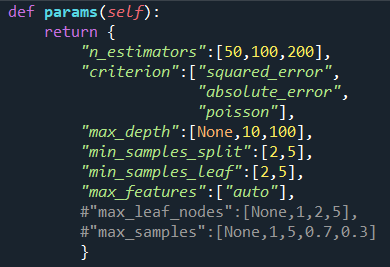
Also we applied the algorithm of feature selection for this model, howver in this case we have an interesting behaviour: the features selected by the algorithm are not always the same. We can divide this set of features in two parts:

* Features always present in the set, that are O3, the temperature and NO2;
* Features not always present, that are the others.

This happens because this model is randomic, so it behaves differently every time. For this reason, we considered and evalueting three runs of this model.

### Tuning

In order to get the best performances, we use the values we can see in the function below.

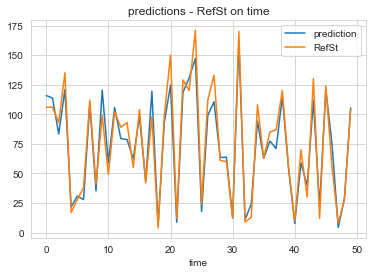
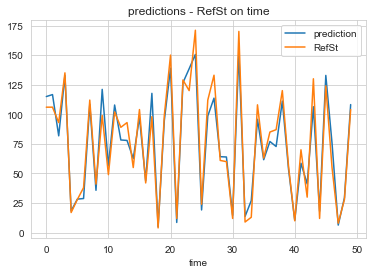
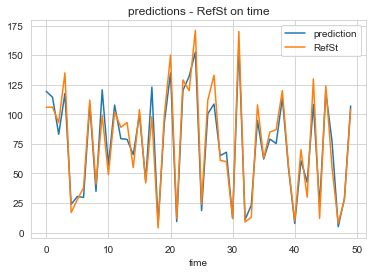


We can see the best values in the following table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Run | #estimators | criterion | Max depth | Min sample split | Min sample leaf | Max features |
| 1 | 200 | Squared error | 100 | 2 | 2 | auto |
| 2 | 100 | Squared error | 100 | 5 | 5 | auto |
| 3 | 100 | Absolute error | 10 | 2 | 2 | auto |

### Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run | Features | R2 | RMSE | MAE |
| 1 | O3, Temp, SO2, NO2 | 0.9389 | 9.947 | 7.591 |
| 2 | O3, Temp, NO, RelHum, NO2 | 0.942 | 9.719 | 7.487 |
| 3 | O3, Temp, NO, SO2,NO2 | 0.942 | 9.723 | 7.486 |



Basing on the three executions about, we can conclude we have a nice model fitting almost perfectly the reference station. Moreover, this model is better than the linear ones for the calibration, in fact the R2 value of this model are greater and, comparing the plots, we can see that these ones fit better.

So, event through the randomness, the random forest model result the best choice for the sensor calibration, but the linear ones and the knn one remains good choices in any case, in fact the differences between them are very little.

## Support vector regression

The last model we use for the calibration is the support vector regression, a model predicting the values building an hyperplane and, in base a certain spacing from the hyperplane, the points will be regretted. The concepts allow a linear regression, but we can regret non-linearly using a kernel. For our calibration we consider the radial basis function as kernel. The support vector regression has the following hyperparameters:

* Gamma is the kernel coefficient for certain types of kernel like the radial basis function;
* C is the regularization factor;
* Epsilon is the range of spacing between the hyperplane to the outside;

Regarding the feateare selection, the best features are O3, the temperature and the humidity, curiosly these are the features with the best squared correlation.

### Tuning

To do the tuning we used the function below:

Immagine che contiene testo, interni, screenshot

Descrizione generata automaticamente

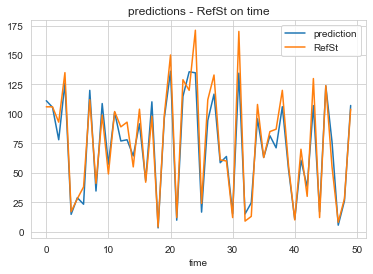
We got the following best values:

* C is equals to 1000;
* Epsilon is equals to 0.001;
* Gamma is scaled, it mean that the value is computed in this way:

1/(#features \* X.var())

### Results

|  |  |  |
| --- | --- | --- |
| R2 | RMSE | MAE |
| 0.928 | 10.88 | 7.876 |



From the results, we can see this is a very good model, in fact the plot show clearly how the predictions fits very well the reference station. Comparing this model with the previouses, this is one of the best and it should be worse than the random forest and better than the linear ones. In any case the differences between this model and the best ones are not significant, so we can use this problem for the sensor calibration without problems.

Fourth homework of TOML

# Introduction

This project consists in the analysis of a feed forward neural network with three layers in a classification problem. Each neuron in the neural network use the ReLU function to compute the output, except the one in the output layer which use the sigmoid.

Regarding the dataset, each point is formed by three components, the first two are generated randomly, the third one is set to 1 and represents the bias.

How it works the classification? Given a pair of radiuses, a point is in the class 1 if its distance from the origin is between these two values, otherwise it is in the class 0.

The model computes the weights (that is random at the beginning) using a gradient descent method with momentum, also there is the application of the backpropagation in order to increase the performances and get the results faster. The momentum is defined by a value alpha we pass to the model as a parameter.

Another parameter of the model is the learning rate, it is set to a constant value for simplify the project.

Last but not least, we try to implement the algorithm of stohastic gradient descent, in the next point we’ll show how the results differ using this algorithm or not.

# The code

To perform this project we implemented a class NeuralNetwork, it takes some parameters, make the classification, plots the results and compute the accuracy.

The method computer takes the number of iteration and the momentum to train and test the neural network at each iteration. Moreover it does the backpropagation and plots the loss functions if the user set the flag “plot” to true.

Immagine che contiene testo

Descrizione generata automaticamente

There is also a method called computeStohastic that does the same things but with the stohastic gradient descent.

The method computeAccuracy is the one which calculate the accuracy of the neural network. To do this, we use a confusion matrix (replaced with four variables for efficience) and we increment the right cells comparing the predicted label with the expected one. After this, we compute the accuracy with this formula:

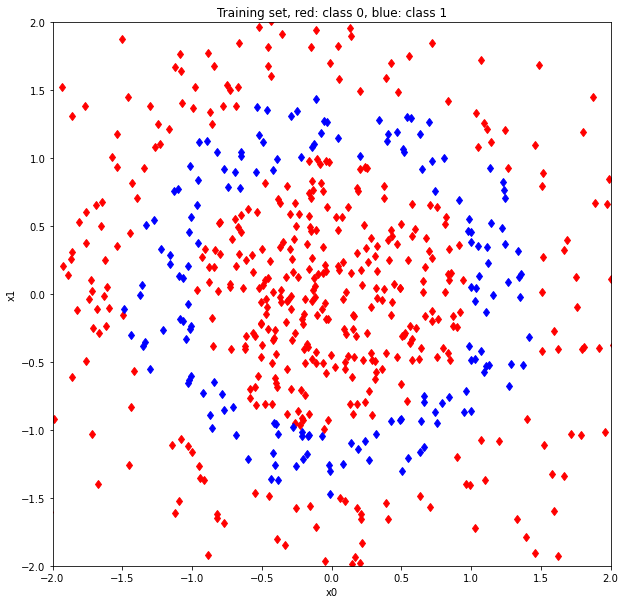
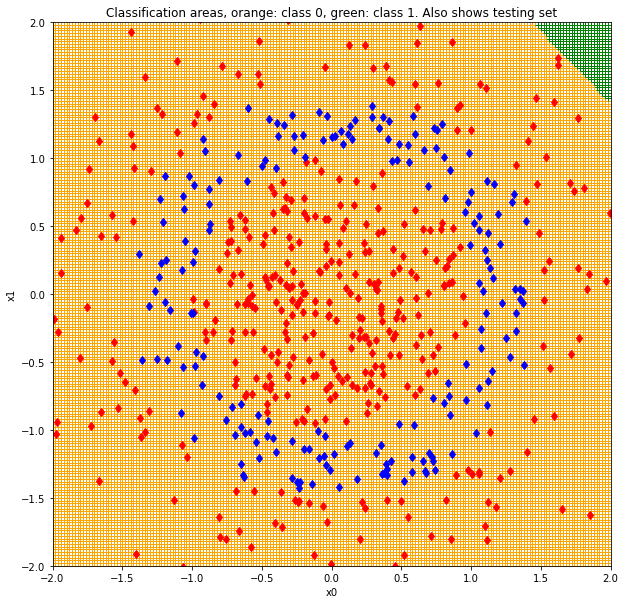
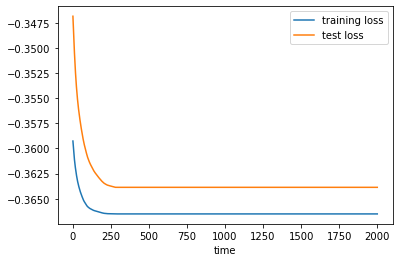
Accuracy= (Tp+Tn)/Total

# Case of underfitting

A situation of underfitting happens when the number of neurons in the hidden layer is very low. So, to perform this purpose, we set the size of the batch to 640 and we varied the number of neuron in the hiddens layers, precisely we considered the values 1, 3 and 5.

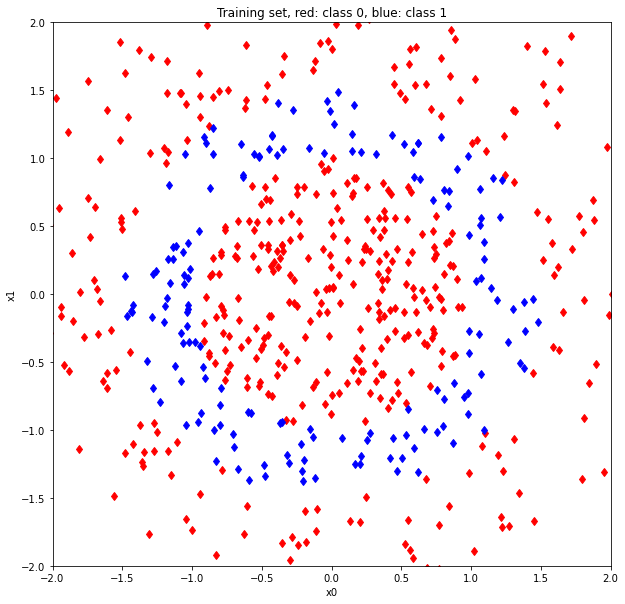
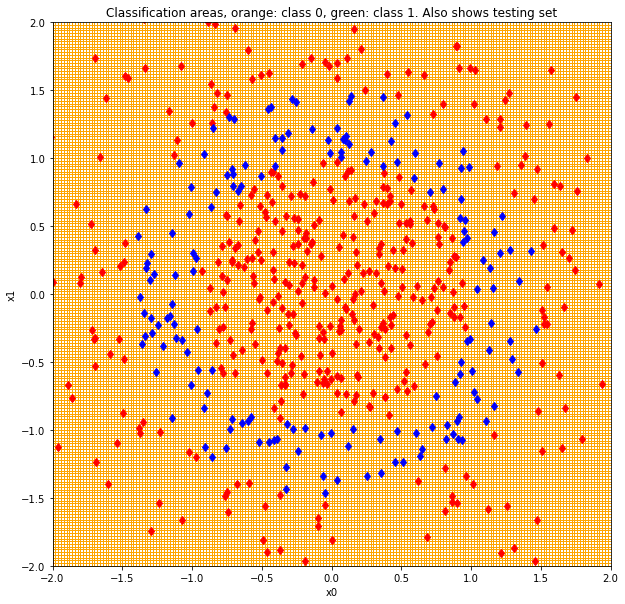
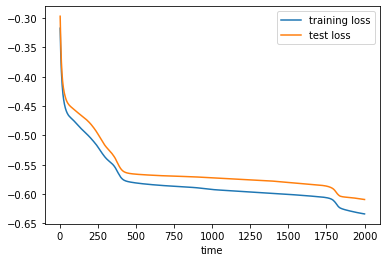
## Results with the normal gradient descent

Considering one neuron in the hidden layer, the model have an accuracy of the 73%, it seems a good results but let’s see the plot:

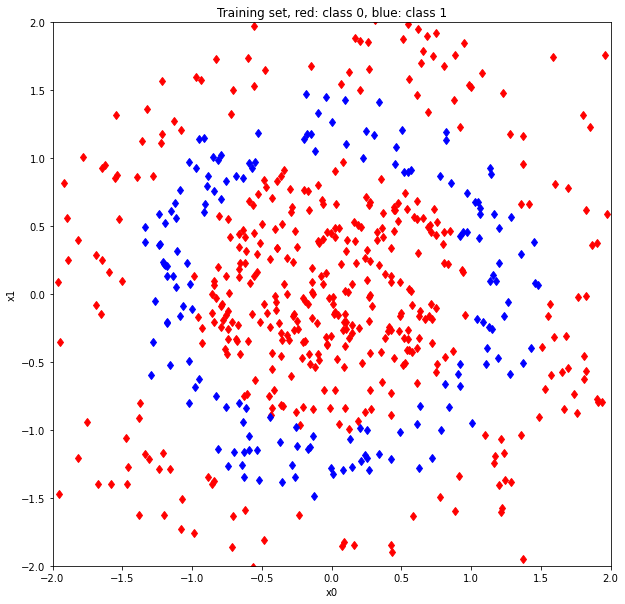
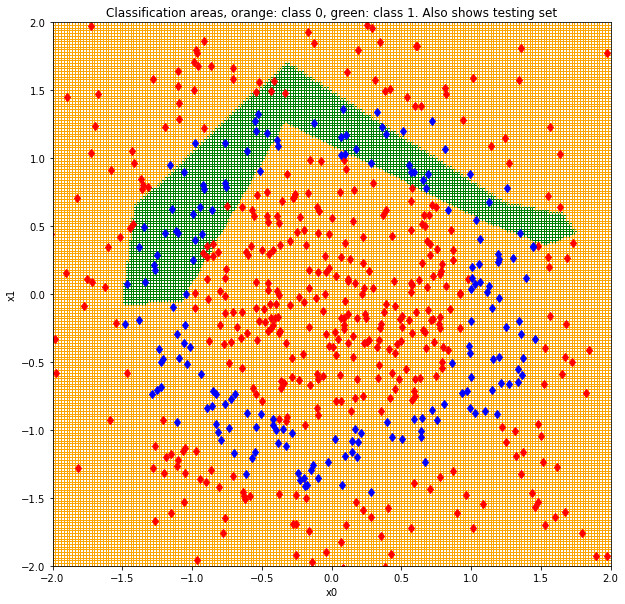
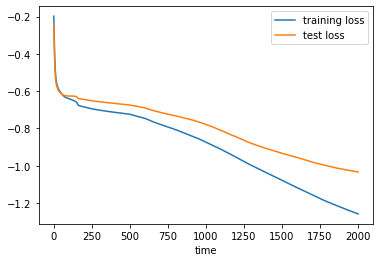
From these plots, we can conclude the model doesn’t classify correctly, in fact there are no green areas specifying the points in the class 1. So this is a clear case of underfitting.

With three neurons, we got an accuracy of about the 71%, this lack of performances respect the first model is principally caused by the randomness, but let’s see the plots:

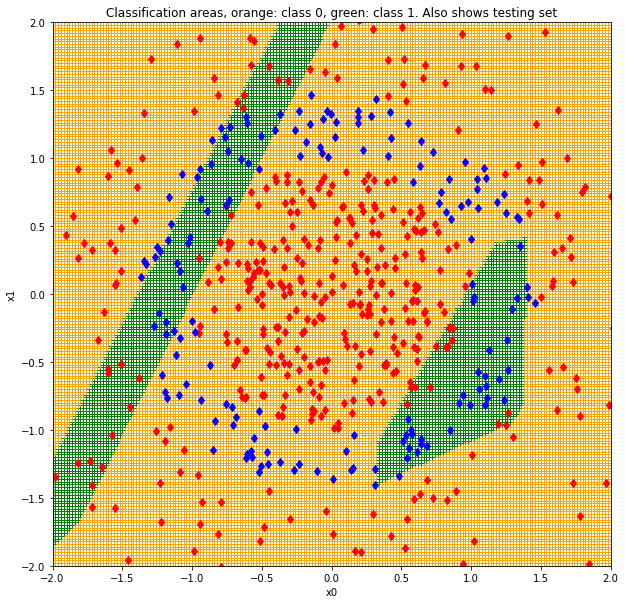
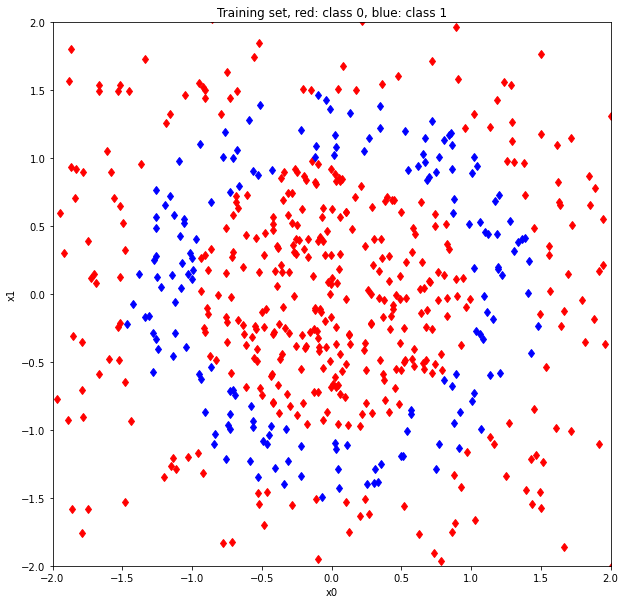
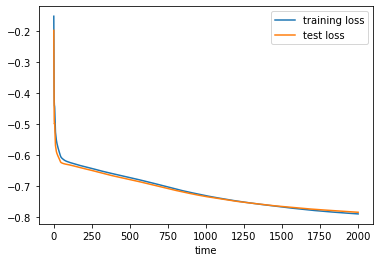
  

As we can see, the situation is the same of the first case but the plot of the loss function has a different shape, something is happening!

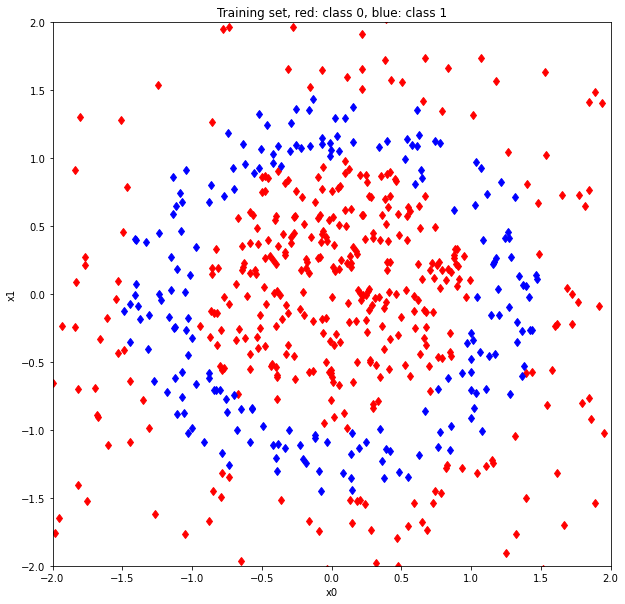
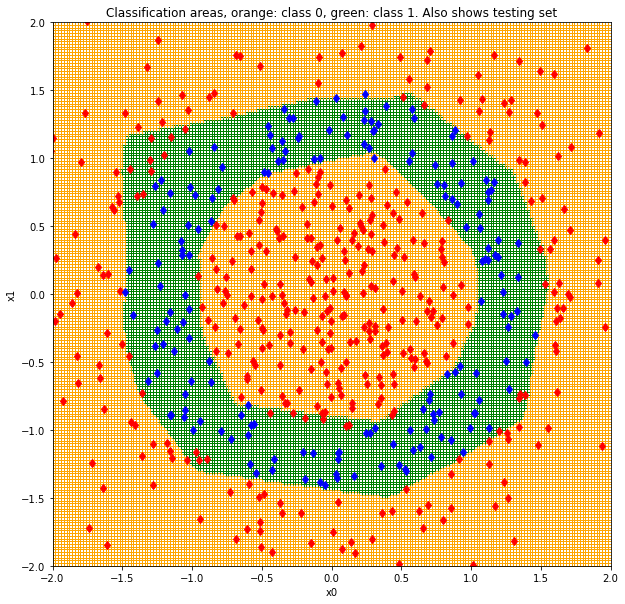
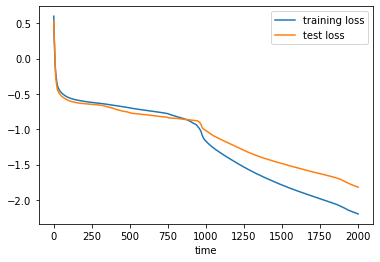
The neural network with five neuron in the hidden layer give an accuracy of the 80%. From the plots we can see that the situation begin to change, in fact there is a green spot covering more or less half of the point of the class 1.

Considering 7 neurons in the hidden layer, the situation is different, in fact we have more spots in the plot, so the classification become better and better. However, these spots are not precise, in fact we have an accuracy of the 75%, an unimprovement respect the previous case.

The neural network with nine neurons seem a balanced situation between underfitting and overfitting, but in the runs there are situation where the model underfits. In this case, with an accuracy of the 73%, we got a plot where the class 1 is clearly divided by the class 0, with some errors.

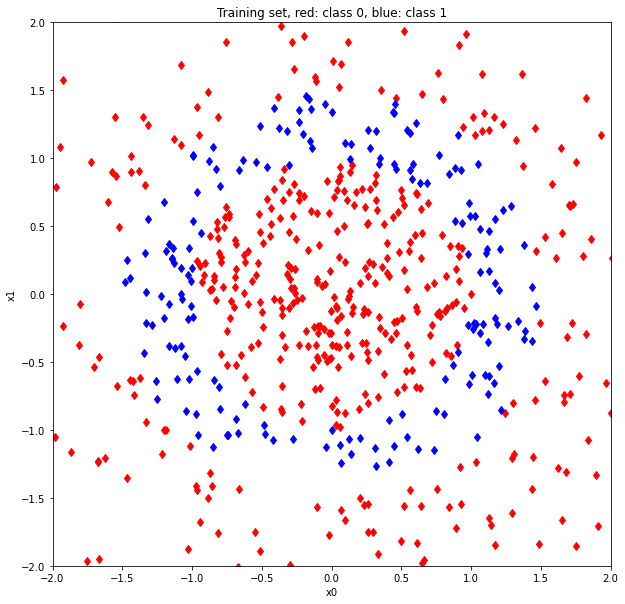
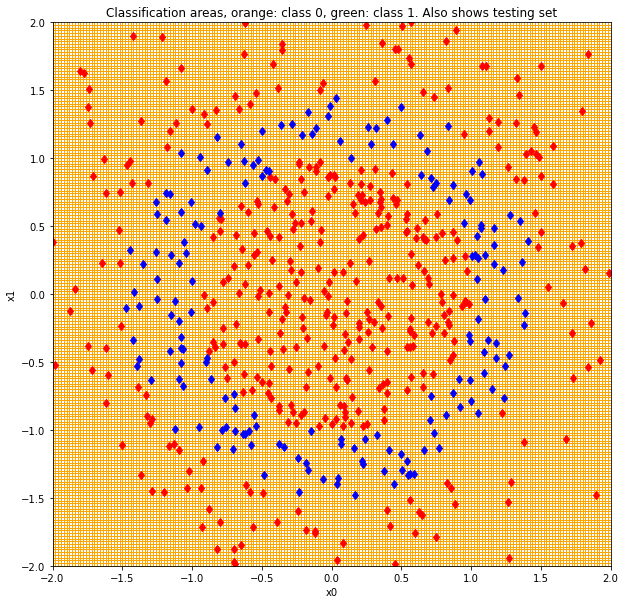
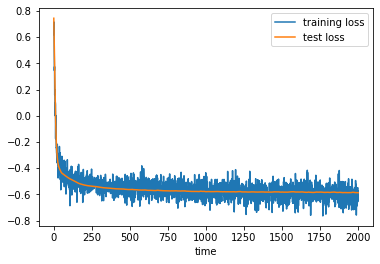
 

### Results with the stohastic gradient descent

Now we analyze the underfitting in the stohastic gradient descent using the same numbers of neurons.

For one and three neurons, the situations doesn’t change, so let’s see the other cases.

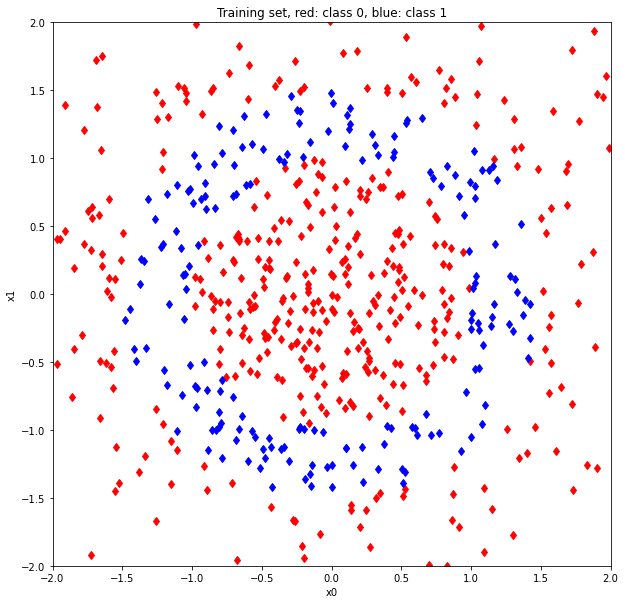
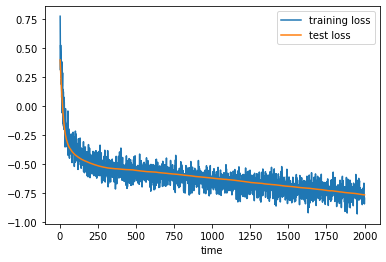
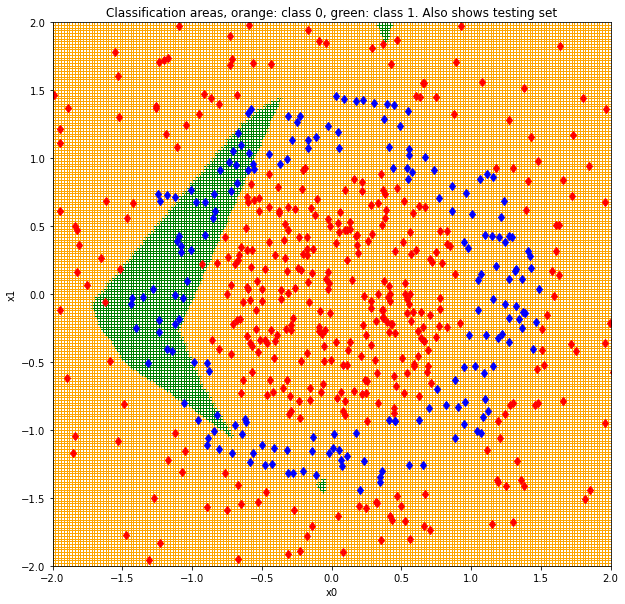
With five neurons we have a case of underfitting with the accuracy of the 73%, in any case there are no green spots in the plot, so we have a clear case of underfitting.

Moreover, the loss plot presents some noise, this happens because the stohastic gradient descent take a minibatch of random points, in fact in each epoch we use a different subset of points that influence the training and the predictions.

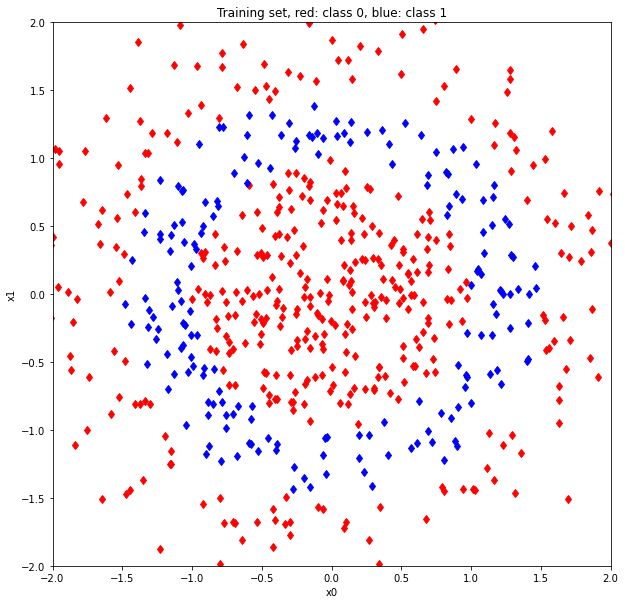
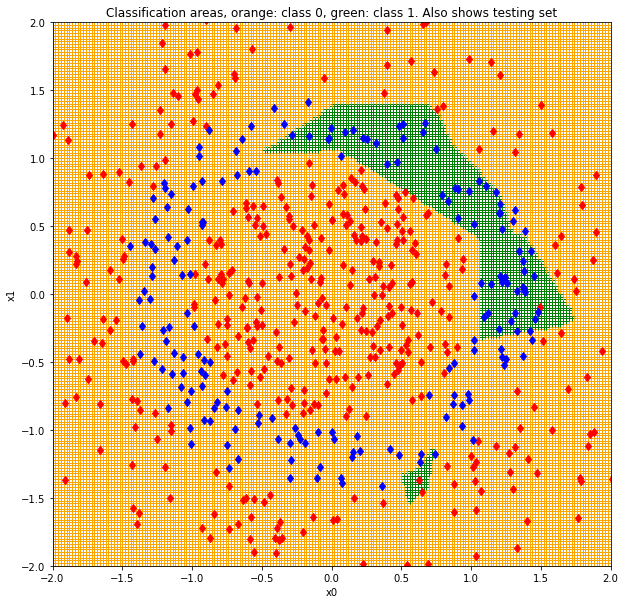
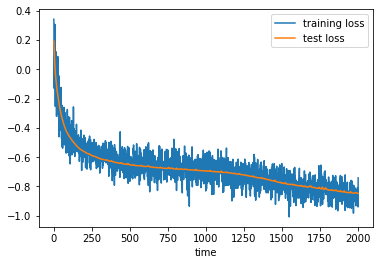
With an accuracy of the 76%, the use of seven neurons gives a situation similar to the previous, in fact we can see from the plot that the green spot is more or less in the same location, but the shape is different.

Regarding the loss plot, we can see that the noise is decreased, so the reseults should be more precise.

Last but not list, the use of nine neurons in the hidden layer gives an interesting result: from the plot we can see a situation of underfitting where there is one green spot, but this one is in a different location respect to the previouses. Moreover, the shape seems more precise respect to the previouses, so we can conclude that the higher is the number of neurons in the hidden layer, the higher is the precision of the shape. In any case we have to consider the randomness of the model, in fact different run could give better results.

Also in this case the loss plots present noise but it is less than the previouses.

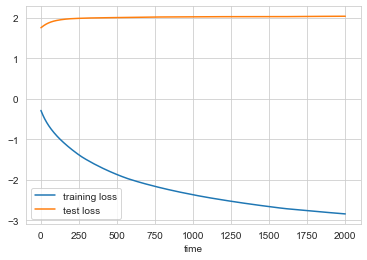
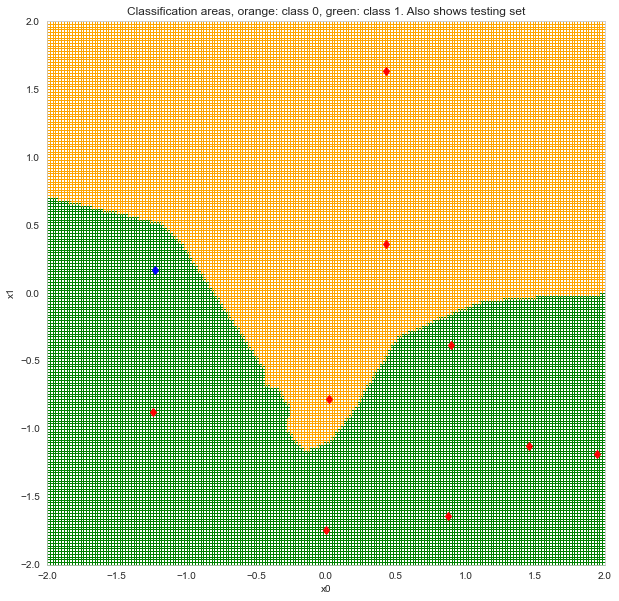
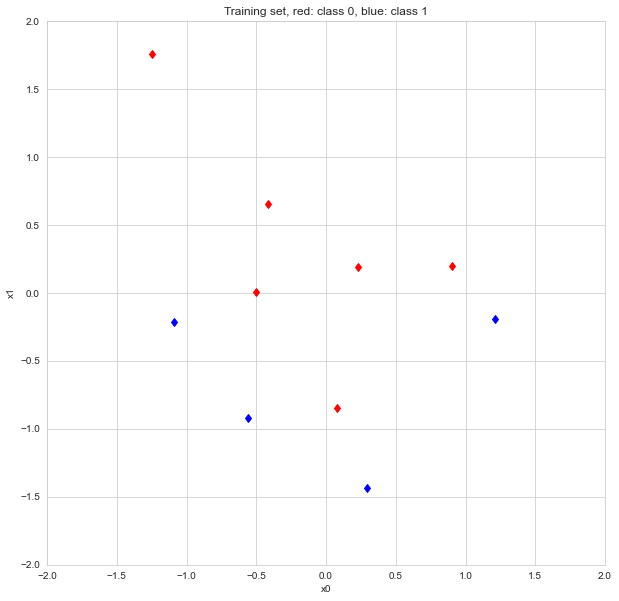
 

# Case of overfitting

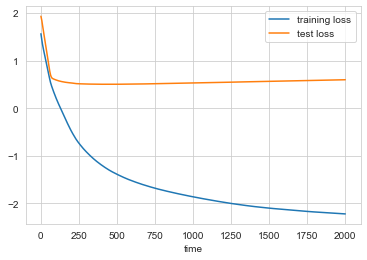
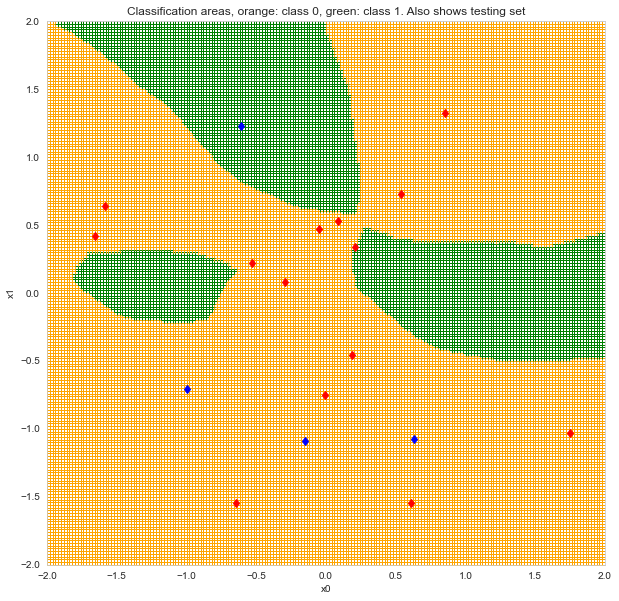
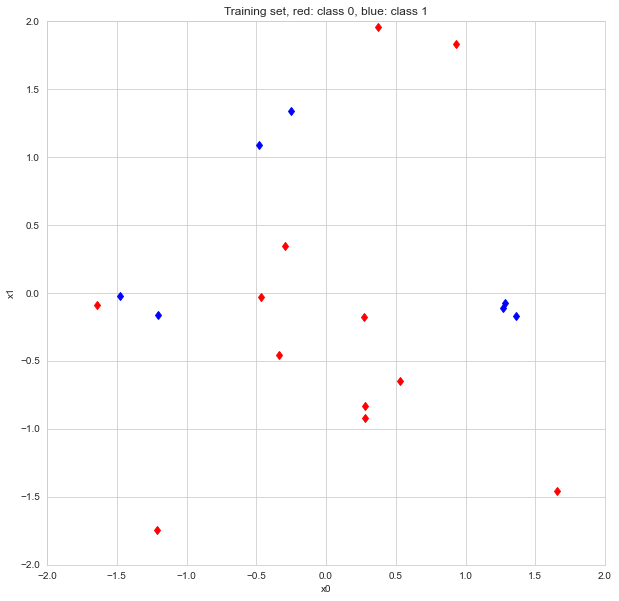
To analyze the overfittinh of the neural network, we do the opposite of the underfitting: we vary the dimension of the batch and we fix the number of neurons in the hidden layer, precisely we used 20 neurons. To perform this part, we use the following values: 10, 20, 50, 100 and 200.

Considering a batch size of 10 points, the neural network tends to draw green spots near the class 1 points. In fact we can see in the plots that the training points are only in the low half of the plot, so the neural network learns that the class 1 points are only in this part and specialize itself, this is a problem because a test point could be classified bad! In fact, we can see in the loss plot how the specialization of the neural network causes a great difference between training error and test error.

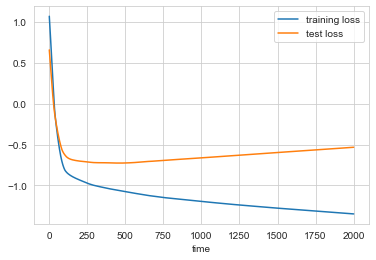
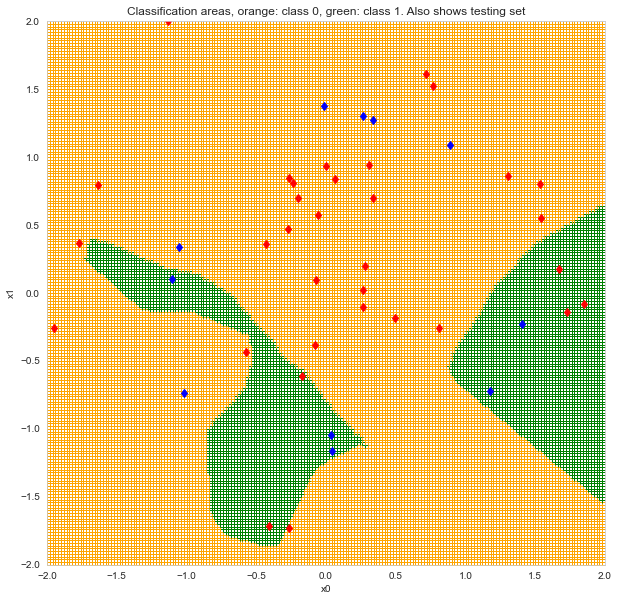
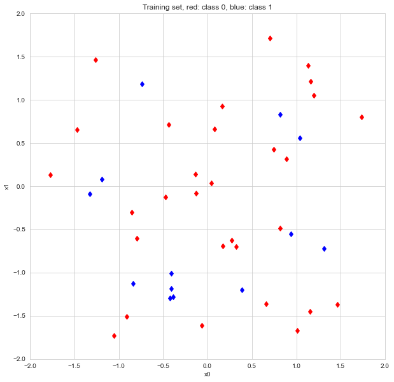
## Results with the normal gradient descent



With batch size of 20 point, we have a situation similar to the previous one, in fact we can see from the first plot that the some points are very near, so the neural network draw spots only in these zones. Because of this, this is a clear case of overfitting and, as we can see in the second plot, it cause a bad classification of some test points. Regarding the loss plot, we have an improment of accuracy respect to the previous case, passing from 0.4 to 0.8.

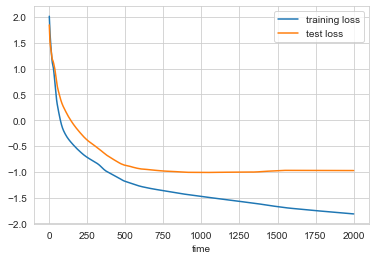
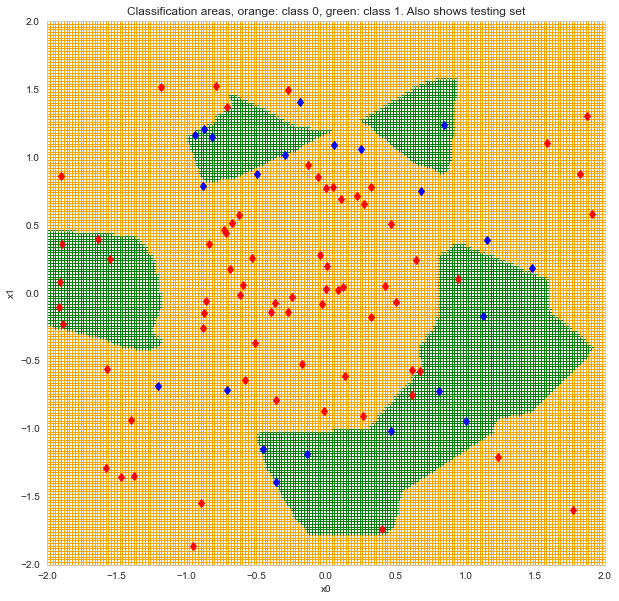
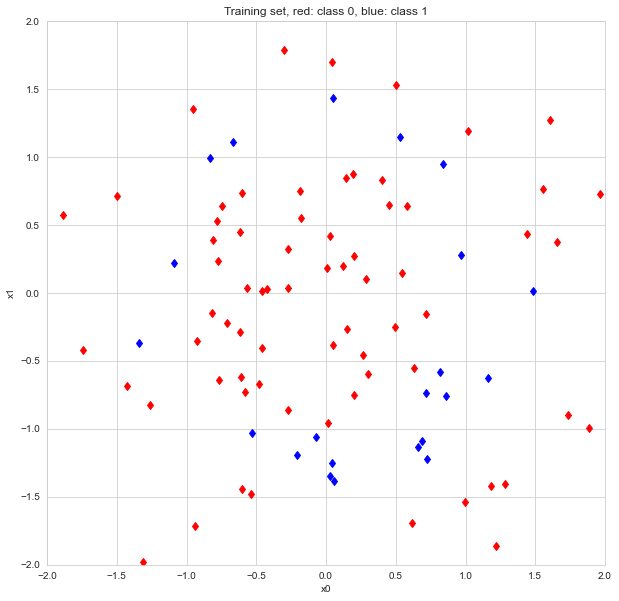


Using a batch size of 50 points, the situation begin to generalize, in fact we can see from the first plot that the training points begin to form a sort of circle. However the situation is not almost changed because there is a bad classification of some test points, as we can see in the second plot. In fact we have an accuracy of the 76% because of this.

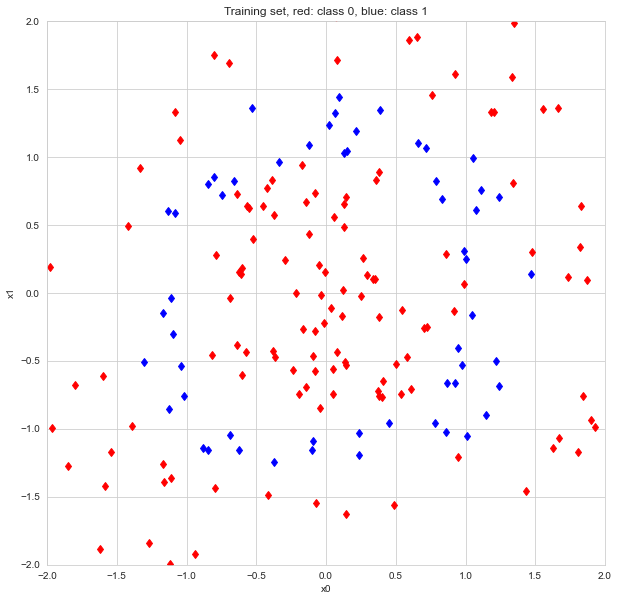
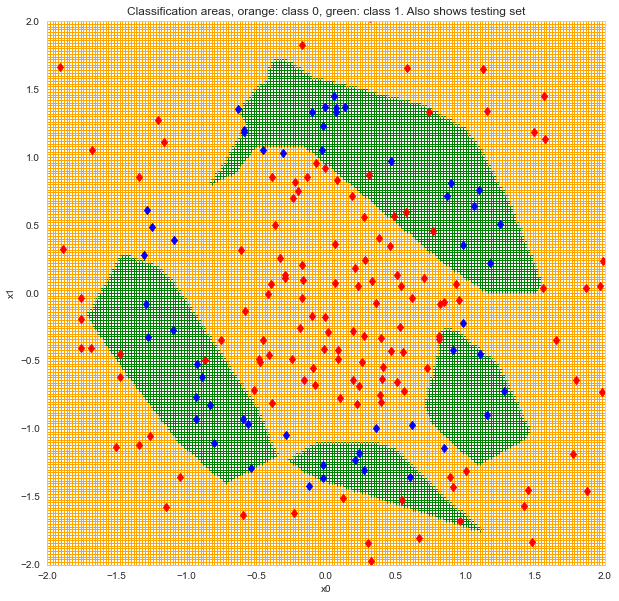
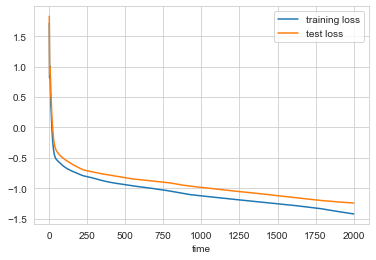


Incrementing the batch size to 100, the situation improved. In fact, even through the overfitting, the green spots begin to draw a circle, this is a sign that the neural network become more and more generalized!

Thanks to this, we have an improvement of the accuracy, passing to the 79% and the difference between training loss and test loss become more and more minor.



Last but not least, the batch size of 200 is an improvement of the previous one! In fact, as we can see in the plots, the green spots begin to be a circle an this improves the accuracy to the 89%! Because of this we can see in the loss plot how the training loss and the test loss become more similar.

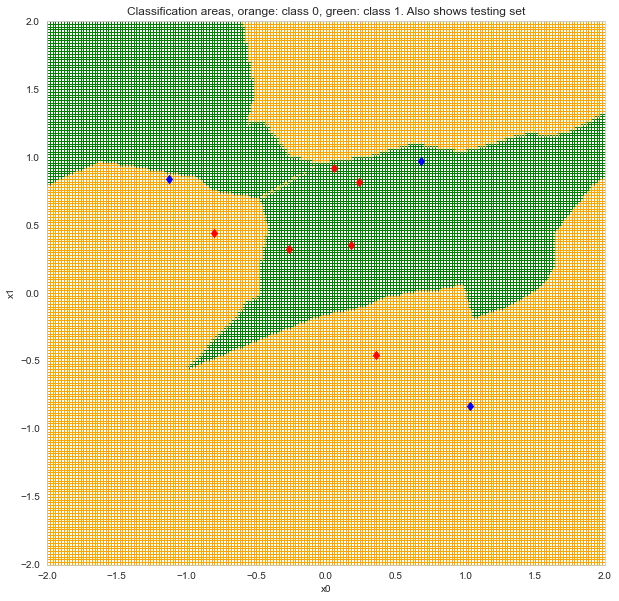
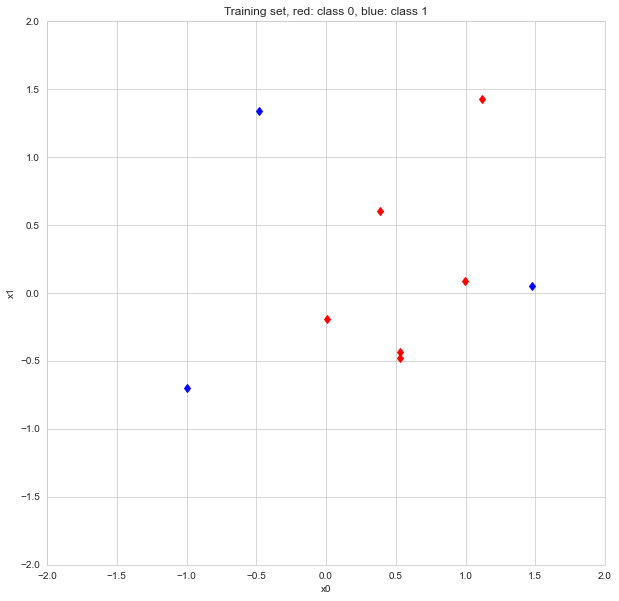
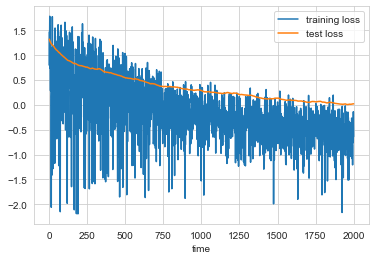
  

## Results with the stohastic gradient descent

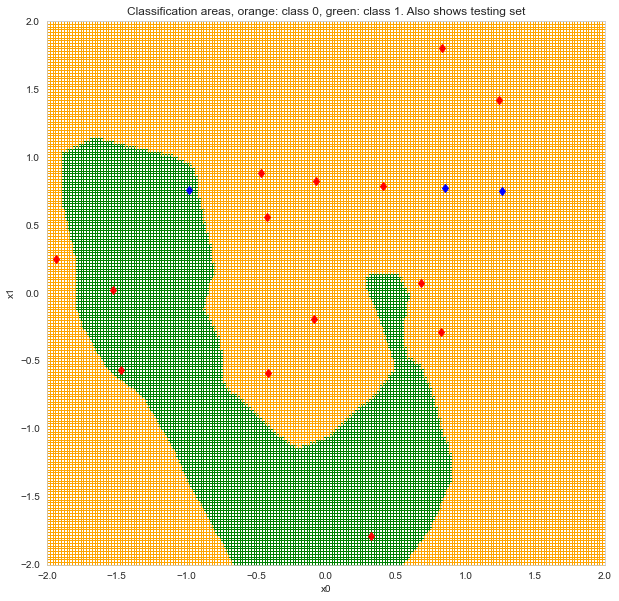
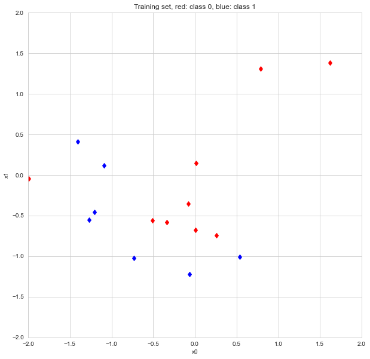
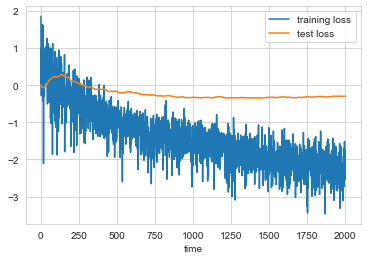
We tested the overfitting using the stohastic gradient descent on the same values of the previous points. Also we consider the size of the minibatch as the half of the principal one.

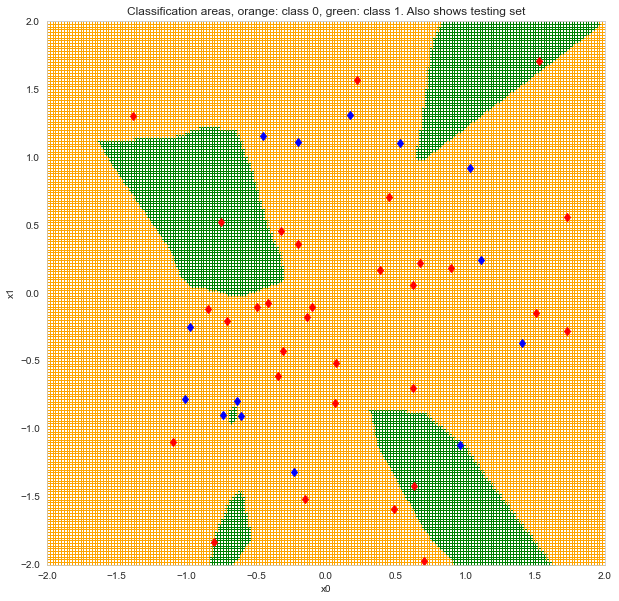
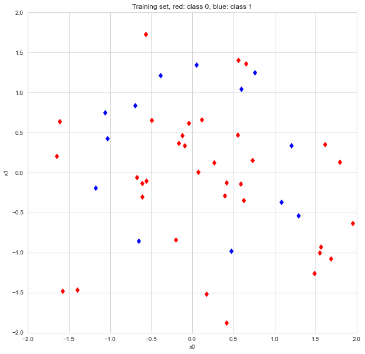
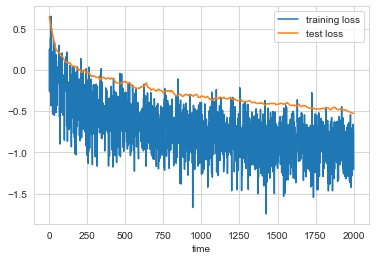
With a batch size of 10 points, we have the situation we can see in the plots. So, there no enough point and the model specify itself as we can see in the second plot, this cause an accuracy of the 40%.

Because of the randomness of the points, the model presents some noise in phase of training.

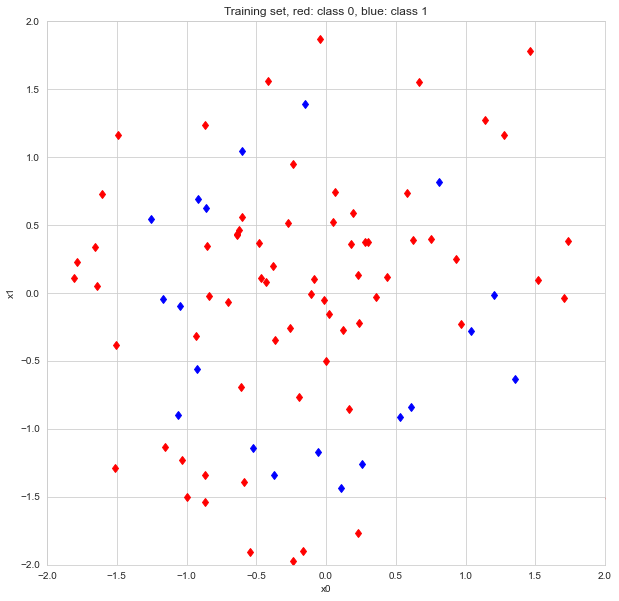
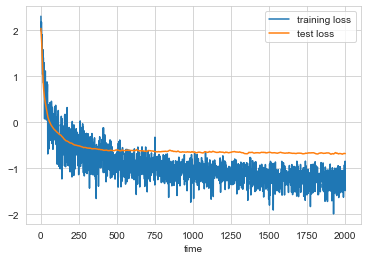
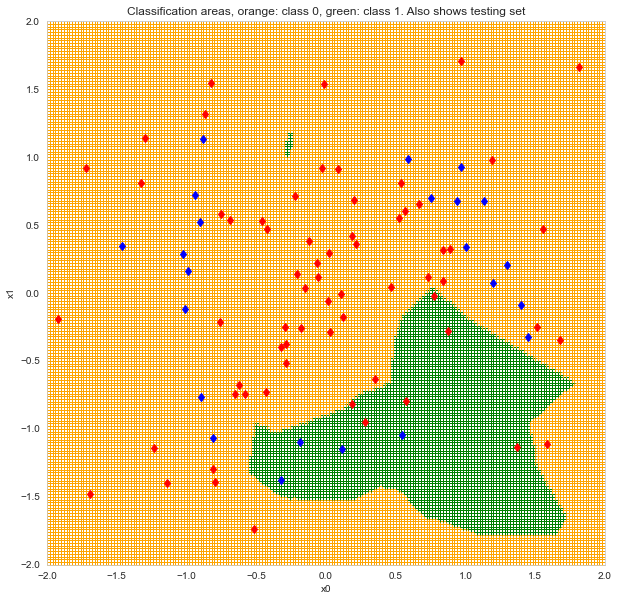
 

Considering a batch size of 20, the overfitting is clearly visible, in fact all the training points of the class one are in a specific area of the plot, so the green spot is drown there. In any case, we have an improvement of the accuracy, passing to the 75% and a decrement of the complessive noise during the training.

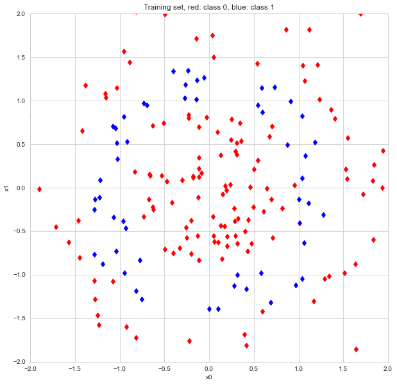
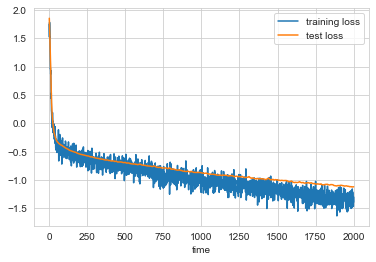
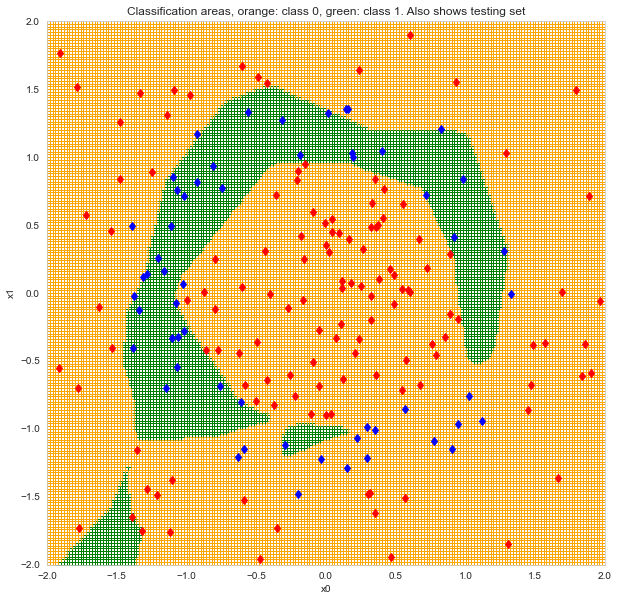
   
Passing to a batch size of 50 points, we can see the training points of the class 1 begin to form a circle, however the neural network is not able to identify it an so it draw green spots only where the points are closer.

The batch of 100 points is a case of overfitting where the training point of the class 1 are more concentrated in the low half of the plot, if fact the model draw the green spot there.

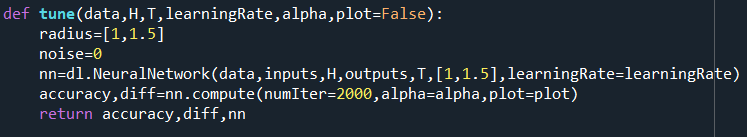
Last but not least, the batch of 200 points is interesting because the model begins to generalize. In fact we can see from the plots that the green spots begin to be a circle, so the prediction wil be more and more correct. To confirm this, we have an accuracy of the 87.5% and a minor difference between training loss and test loss.

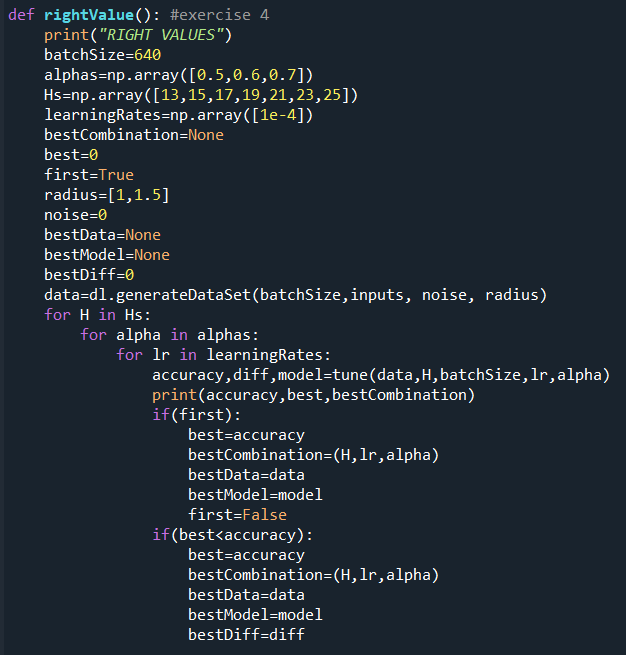


# Right values for the model

In order to find the correct values of neurons, alpha and learning rate, we did a tune of the neural network and we chose the model which increases the accuracy.

We found the rights values using these functions:





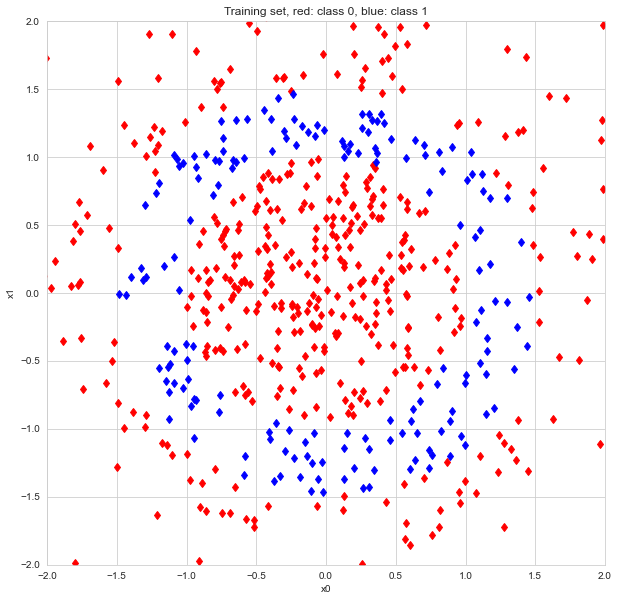
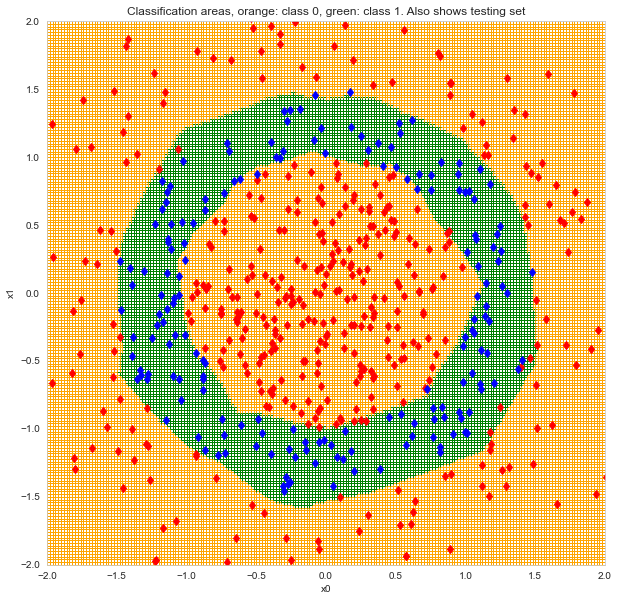
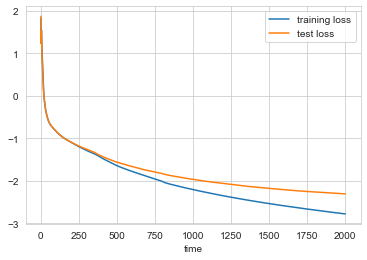
## Results with the normal gradient descent

The result varies because of the randomness of the weights and of the input data, in any case we found these values with some executions:

* A learning rate of 10^3;
* A momentum around 0.5 and 0.7;
* A number of neurons between 17 and 19;

The following plots regards a model with 19 neurons in the hidden layer, a learning rate of 10^3 and momentum of 0.7.

In these plots we can see how the plot fits very well, the green part forms a circle, that means it follows the training points of the class one very well. In this way we have the besta value of accuracy, that is the 95%!

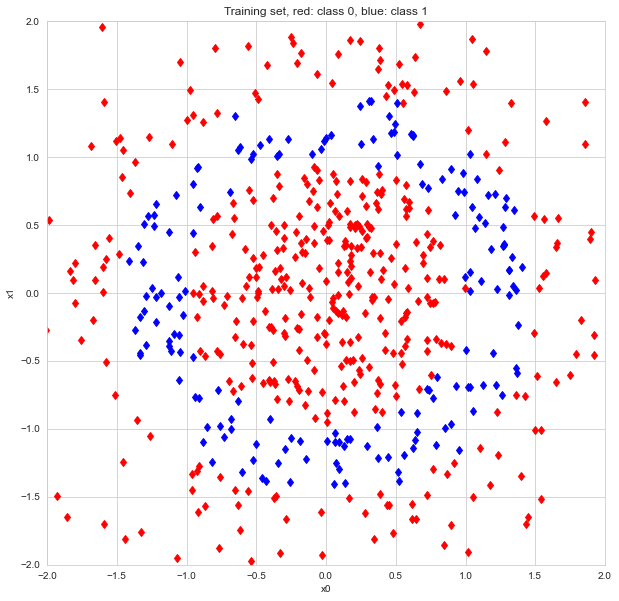
## Results with the stohastic gradient descent

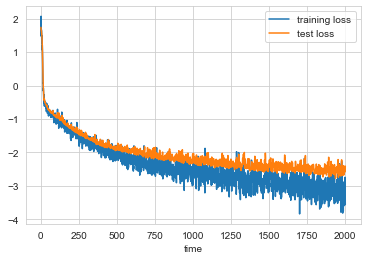
We redo the same experiment with the stohastic gradient descent, using the same values for the tuning, we only add 10^-3 for the momentum in order to get more various models.

With some executions, we can conclude that:

* The best number of neurons is about 18 or 19;
* The best momentum is 0.6;
* The best learning rate is 10^-3

The plots below are about a model with 19 neurons in the hidden layer, a momentum of 0.6 and a learning rate of 10^-3, here we can see how the model fits very well the training points in the class 1, because of this we have a circle-shaped green spot and an accuracy of the 96%! The loss plot is noisy because of the randomness of this method, in any case we can see that the training loss and the test loss are quite similar.

 Immagine che contiene mappa

Descrizione generata automaticamente

# Problems

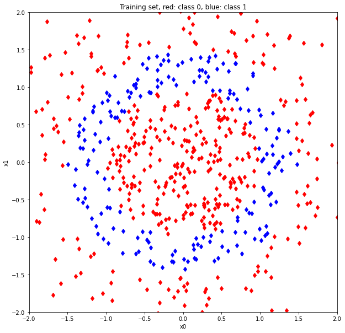
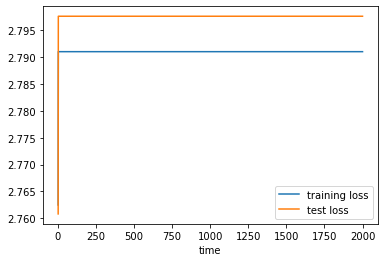
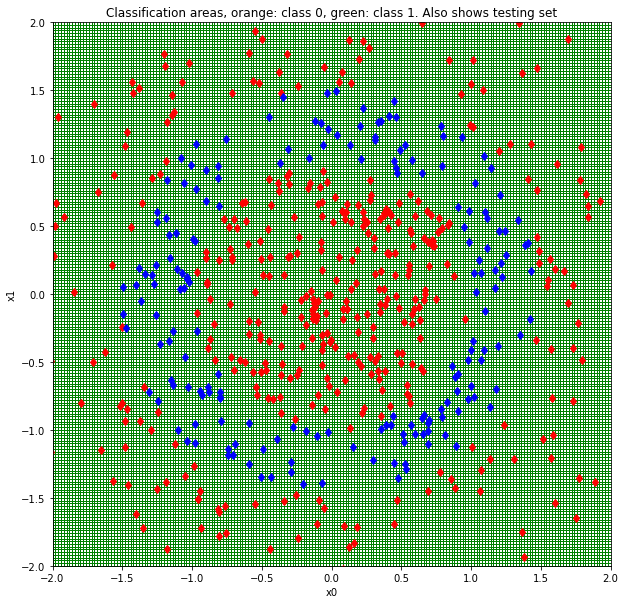
To conclude this report, we analyze the situations where there are some problems, they occure when the number of neurons in the hidden layer is very high. In order to perform this point, we use the following values as number of neurons: 100, 320, 640 and 1280.

Regarding the batch size, we fix it at 640, as in the previous points.

## Results with the normal gradient descent

In all the experimentation we did, the loss function as an initial large drop or escalation and then it becomes constant, this is caused by the gradient that belongs to zero when there are too many neurons in the hidden layer.

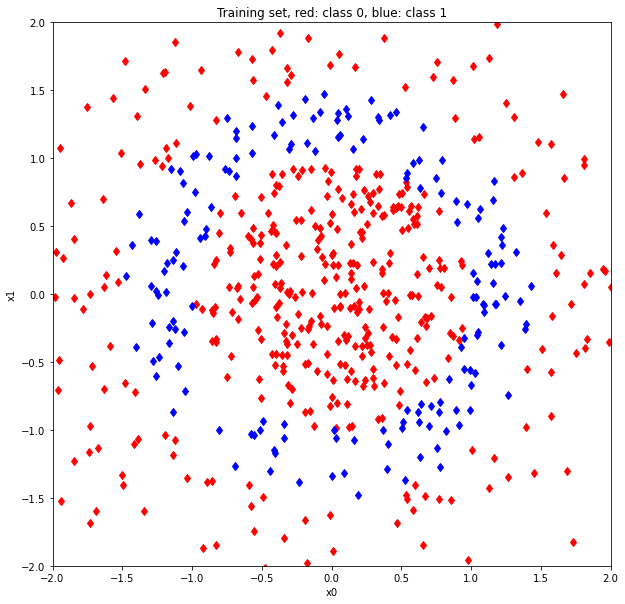
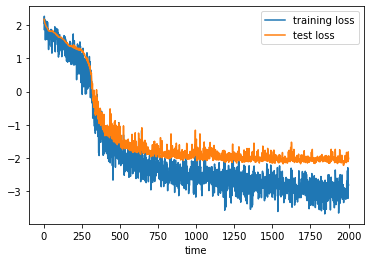
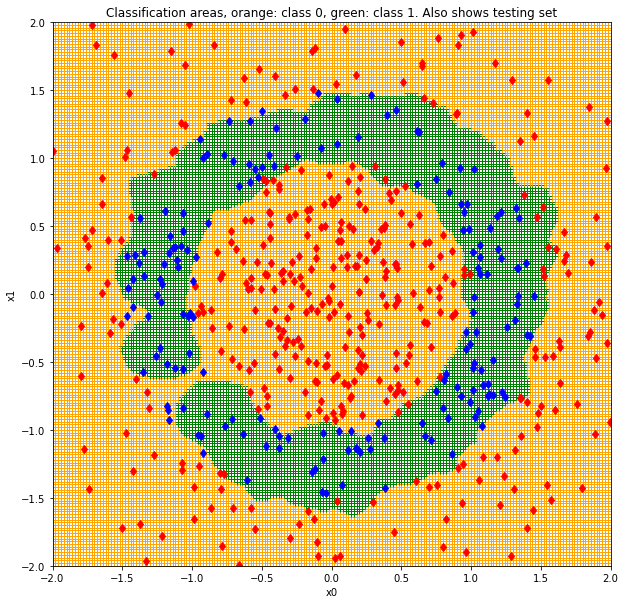
Moreover, there is a lack of performances, we can see it in the plots below:



## Results with the stohastic gradient descent

Repeiting the experiment with the stohastic gradient descent using a minibatch of 106 , we have some happenings:

* With the values 100 and 320, the model fits drawing the classic circle, however it is not a perfect circle, in fact in some parts could be opened and the border is very articulated.
* Also in these cases the loss functions are noisy but they haven’t the same behaviour of the previous case;
* In some cases the gradient don’t belong to zero, so the lost functions should not be constant;
* In any case, there are values that don’t improve the model.

## 

# Notes

All the code is available here: https://github.com/MarcoCarry97/Topic-On-Optimization-And-Machine-Learning