NEURAL NETWORKS AND GENETIC ALGORITHMS

FERTILITY DATA SET

REGRESSION

BRATUCU ANA MARIA

GROUP 1231EA

**Multi-layer Perceptron Method**

The chosen Fertility data set reveals information about the fertility criteria (socio-demographic data, environmental factors, health status, and life habits) on 100 subjects. Given this information, we have to predict the fertility diagnosis (altered or normal). The features represented by the information that affect fertility like age or traumas, all represented by values between -1 and 1 (independent variables) and the class attribute (dependent variable), that consists of a column of strings. The healthier and younger they are, the more chances are to have normal fertility.

Data set link: http://archive.ics.uci.edu/ml/datasets/Fertility

Attribute details:

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Season analysis was performed in | numerical | winter (-1), spring (-0.33), summer (0.33), fall (1) |
| Age | numerical | age span of 18-36 represented as numbers between 0 and 1 |
| Childish diseases | numerical | yes (0), no (1) |
| Surgical intervention | numerical | yes (0), no (1) |
| High fevers in the last year | numerical | less than 3 months ago (-1), more than 3 months ago (0), no (1) |
| Frequency of alcohol consumption | numerical | several times a day, every day, several times a week, once a week, hardly ever or never on a scale from 0 to 1 |
| Smoking habit | numerical | never (-1), occasional (0), daily (1) |
| Number of hours spent sitting per day | numerical | in a range from 0 to 1, where 0 is “no time spent sitting” and 1 is “all time spent sitting” |
| Diagnosis | string | normal (N), altered (O) |

Given the type of the data set, this is a problem that must be associated to the regression type of Machine Learning, because it is used to study the effects certain factors have in the contribution of fertility diagnosis. The data is not constant, it can change over time, but by using regression, the computer will still be able to predict the output of future new values by studying and gathering information about the patterns of the training set (supervised learning). So, by understanding, for example, that the age of a person is a meaningful decision factor and plays an important part in the fertility results, the algorithm will be able to predict the fertility of new people, based on their age (and of course the other factors that are presented in the data set).

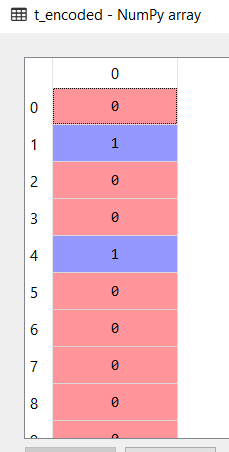
Therefore, we build a neural network. Luckily for us, this set doesn’t contain missing values, so we don’t have to clean the data. For this classification problem’s algorithm, I use:

* the Scikit-learn implementation of the Multi-layer Perceptron regressor (MLPRegressor) to generate my the Perceptron algorithm
* the pandas library for handling the data set (read the .csv file from my computer)
* the matplotlib library to plot the error of the neural network
* the numpy library for the mathematical calculations

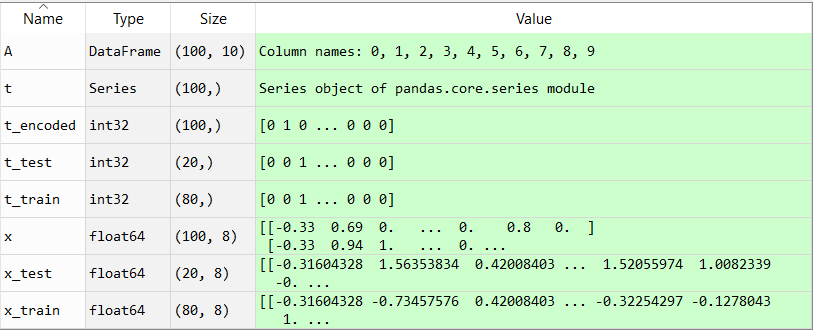
I start off by breaking the balance scale data set into one with features and another one only with the column of the class. Because the class is made up of strings (O and N) I transform it in numerical values in order to work with it:

t\_encoded=le.fit\_transform(t)

therefore, I obtain a list with numbers (0 for N and 1 for O)



In order to avoid overfitting, I split the features data set into two subsets: for the training and for the test (test\_size=0.2 => 20% from the whole set for testing and the rest of 80% for training). This way, when I make the test, I make sure that the network won’t return the right output just because it trained too well and memorized the data it was trained on. If that happens, the performance will decrease on completely new data point.



In order to get the best results, we want to normalize the data, so there is a standard format for it. That way, the training is improved, more accurate and faster. Using sklearn’s method, StandardScaler, I scale the data (training and testing sets):

scaler = StandardScaler()

scaler.fit(x\_train)

x\_train = scaler.transform(x\_train)

x\_test = scaler.transform(x\_test)

Afterwards, I create the MLPRegressor function, build the Multi-layer Perceptron classifier from the training set (x, t) and train the model:

net= MLPRegressor(activation = 'logistic' , max\_iter=1000, verbose=True,

hidden\_layer\_sizes=(15,15,15), random\_state=1)

net.fit(X\_train,y\_train)

* activation=’logistic’ because it is more complex for predictions that are not linear and can view probabilities
* max\_iter=1000 represents the number of steps; the number of iterations doesn’t have to be very big, considering the relatively small data set
* verbose=True is for printing progress messages
* I chose hidden\_layer\_sizes to be (10,10,10) because I want to have 3 layers with 10 neurons each as I have 10 columns in the data set.

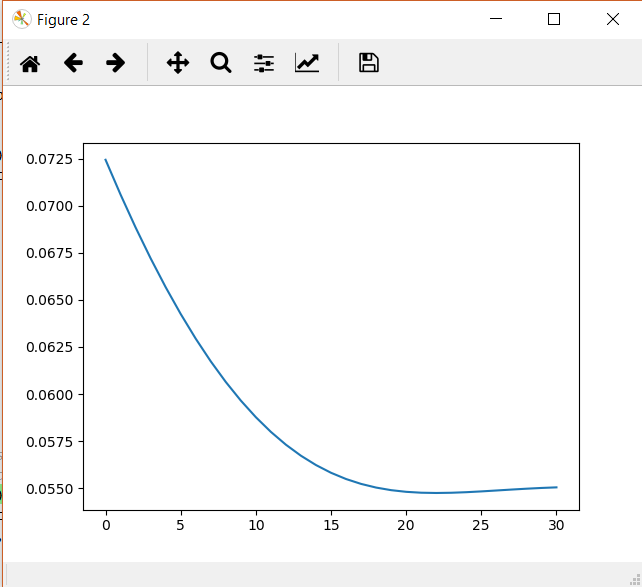
Now that the neural network prepared itself with the training set and learned its patterns, it’s time to predict the outcome for the testing set:

y\_test=net.predict(x\_test)

and compute the loss:

net.loss\_curve\_ , where loss\_curve\_ is a built-in function for the MLPRegressor for finding the cost function of the Perceptron.

Then, I plot it



OK so our loss is decreasing nicely, but it is a bit slow. Out of 31 iterations, it arrives at its lowest value at iteration 15.

Now it’s time to see if the model is good or not, so we calculate the error metrics to evaluate it and see how we can improve it.

I start by computing the mean absolute error for test to find the typical value of the difference between the actual data and the model’s predictions:

err = np.sum(abs(t\_test - y\_test))/len(t\_test)

Next, I compute the mean square error for test where the predictions of the model that differ greatly in comparison with the actual data are punished more, so there will be a bigger value and a greater distinction if there are a high number of errors.

err1 = np.sum(abs(t\_test - y\_test)\*\*2)/len(t\_test)

Let’s also evaluate the root mean squared error which is defined by

mse = np.sum(abs(t\_test - y\_test)\*\*2)

rmse=np.sqrt(mse/len(t\_train))

Another important metric for evaluation is the coefficient of determination R^2 to measure how well the model fits the observations (the distances between the actual and predicted values are small and unbiased) and, also, how well future data is going to be predicted by the algorithm. It has values in the range of (-1,1).

ssr = np.sum((y\_test - t\_test)\*\*2)

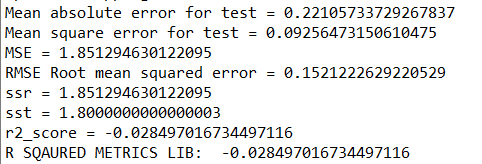
sst = np.sum((t\_test - np.mean(t\_test))\*\*2)

r2\_score1 = 1 - (ssr/sst),

where ssr is the sum of the square of residuals and sst is the total sum of errors if we take the mean of the observed values as the predicted value.

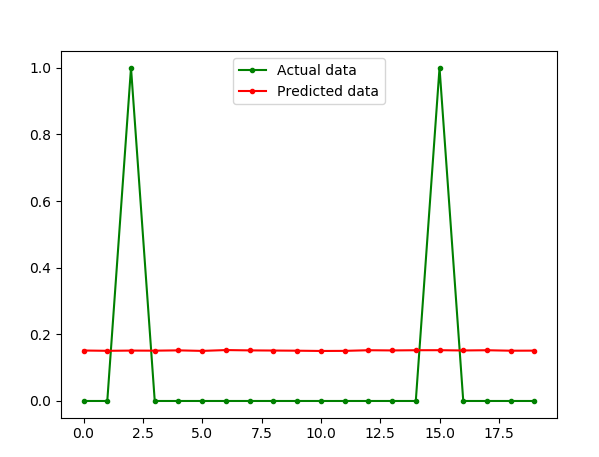
R^2 can also be computed using the sklearn.metrics’ method:

r2\_score(t\_test, y\_test)



As we can observe, the errors calculated have small values, all being very close to 0. The smaller, the errors, the better the predictions, the closer the model is to the actual values and the better it will perform. The problem appears at the R^2 value because it is negative. That means that the algorithm has an increased prediction error of ~2%. Therefore, the predicted values are worse than the actual ones and the model underestimates the fertility values. An explanation for this can be that the data set contains few instances (only 100) and the network doesn’t have enough data to predict well new inputs.

To observe better the difference between the actual data and the predicted one, I plot them both:



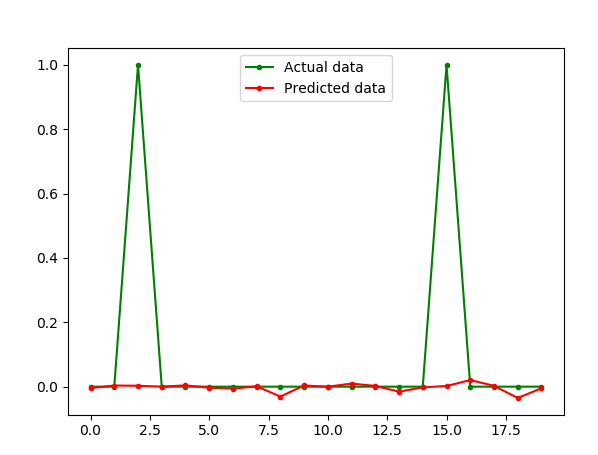
We can see the values of our class from the data set distributed with values of 0 and 1 and the predicted values, which is not very far away from the majority of plotted actual data points and it is the visual confirmation of what we saw before from the small values of the errors.

By adding a parameter, I want to test the changes in the model and hopefully make it more accurate:

net= MLPRegressor(activation = 'logistic',**solver='lbfgs'**, max\_iter=1000,

verbose=True, hidden\_layer\_sizes=(10,10,10), random\_state=1)

* solver='lbfgs' because it performs better on small data sets like mine



From the graph, it is clear that it this setting really does perform better on my data set because the predicted data intersects almost completely the actual data. The only problem is that it misinterprets the cases when the output is 1 because none of the predicted values are even coming close to that number. But the data set contains very few instances when the patient is diagnosed with altered fertility (1) and there are stronger attributes for the class when the patient is diagnosed with normal fertility (0), so this is the explanation for this inconsistency.

**DECISION TREE METHOD**

Another instrument to train a neural network is the Decision Tree. Using supervised learning, it predicts the value of a target by creating a tree like model and using it for learning decision rules indicated from the data features. This is how it is different from the MLPRegressor method presented previously.

For the implementation part, I will use the same libraries and methods as before. I upload the data set and split it into train and testing data, then adding the DecisionTreeRegressor function that will compute the algorithm needed for this method. Moving forward, I build a decision tree regressor from the training set and based on the patterns extracted, the neural network will predict the results for the testing set:

tree =DecisionTreeRegressor(min\_samples\_leaf=30,max\_leaf\_nodes=70,

min\_samples\_split=3,splitter='best',criterion='mse')

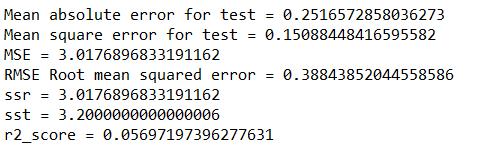
tree.fit(X\_train,y\_train)

predictions = tree.predict(X\_test)

I use the parameters:

* min\_samples\_leaf=30 - the minimum sample size can be fixed to 30 so that’s what I chose because I wanted to restrict the size of sample leaf
* min\_samples\_split=3 for minimum number of samples required to split a node; I chose it to be 3 because I didn’t want the tree to be too constrained as if it was a higher number, it would’ve had to consider more samples at each node
* max\_leaf\_nodes=70 - because my tree is complicated enough and I wanted to simplify it by reducing the number of leaf nodes, expecting e better accuracy
* splitter=’best’ because not all the features in the data set have the same importance in the output, so I want to choose the most relevant information from the data set
* criterion='mse' is by default

Afterwards, I compute the errors just like I did for the first method. This is the result:

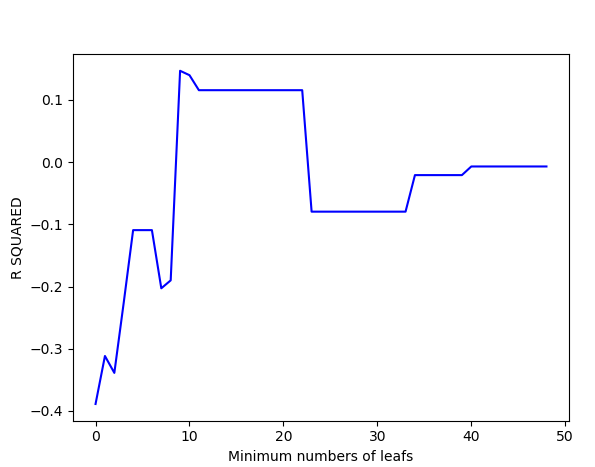


The error values look good, they are small enough to not cause any problems and the most important aspect is that R^2 gave a positive value this time. That means that the algorithm has a decreased prediction error of ~5%. Therefore, the predicted values are better than the actual ones and the model behaves more accurate.

To further test the MLPRegressor’s parameters against the R^2 metric, I change the min\_samples\_leaf to be in a range between 0 and 50 and plot R^2.

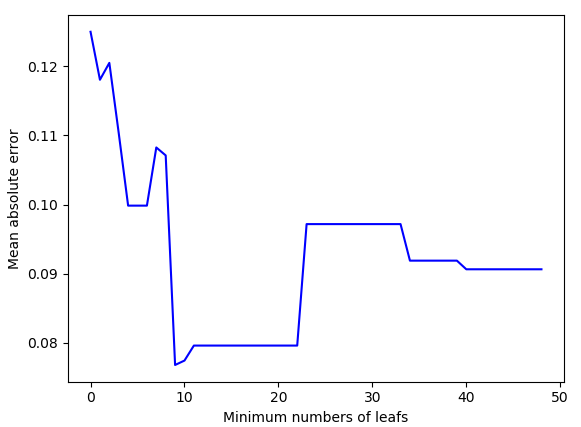
tree = DecisionTreeRegressor(**min\_samples\_leaf=i**,max\_leaf\_nodes=70,

min\_samples\_split=3,splitter='best',criterion='mse')

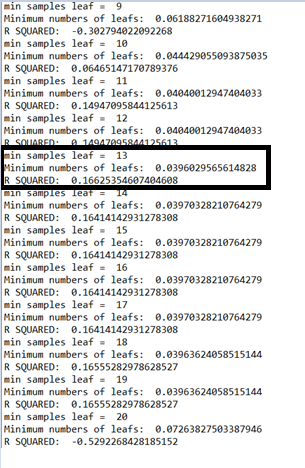


The graph shows R^2 is increasing as the number of leafs is increasing until it reaches 13 leafs, where the maximum value for R^2 is found 0.16, making the model more accurate by 16%. R^2 stays constant for the interval (10, 20) minimum leafs and then is starts to decrease. In translation, the tree needs to be more restricted and have at least 10 minimum leafs, however, making it too complex, the accuracy and quality of the model prediction starts to decrease. If the minimum leaf number is too small (below 10), the coefficient of determination has a negative value, so it’s very bad for the model.

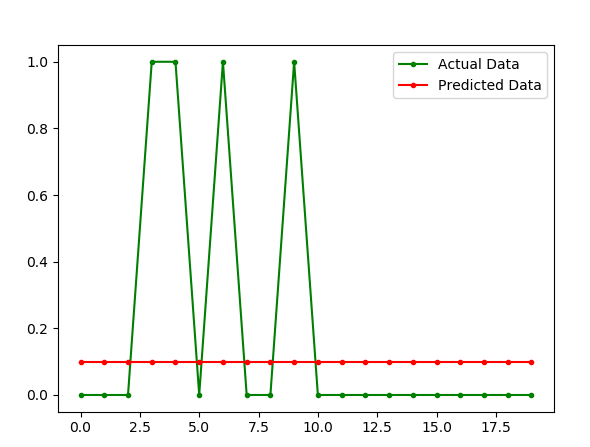
Using the same method of plotting, for the same MLPRegressor parameter (between 0 and 50), I also compute the mean absolute error. These are the results:



Again, I get the best results when the minimum leaf number is 13 (or at least between 10 and 20) because that’s when the mean error is the smallest (0.03). Therefore, the error is inverse proportional to the coefficient of determination.



To observe better the difference between the actual data and the predicted one, I plot them both:



This is exactly like the case from the MLPRegressor. We can see the values of our class from the data set distributed with values of 0 and 1 and the predicted values, which is not very far away from the majority of plotted actual data points and it is the visual confirmation of what we saw before from the small values of the errors. For both this method and for the MLPRegression, this graphs show that our model underfits, the data can’t generalize to new inputs and has a poor performance in predicting the importance of independent data in the fertility diagnosis output.

**Conclusion**

In conclusion, although I split the data set into two subsets (training and test) both of the methods (MLPRegressor and DecisionTreeRegessor) underfit, meaning that the accuracy for predictions is low. Taking this into considerations, I changed the parameters for both the functions and observed the changes in performance. This way, for the DecisionTreeRegressor, I found the best parameter (min\_samples\_leaf = 13) and its best value for improving the model and the lowest error I could achieve. I managed to make the neural network more accurate by 16%. For the MLPRegressor I couldn’t improve it, therefore this method is not very suitable for this data set which has a major problem: it has few instances and that stands in the way of better predictions.

**Bibliography**

1. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
2. https://scikit-learn.org/stable/modules/model\_evaluation.html#regression-metrics
3. https://scikit-learn.org/stable/modules/generated/sklearn.neural\_network.MLPRegressor.html
4. https://www.dataquest.io/blog/understanding-regression-error-metrics/
5. https://www.youtube.com/watch?v=r-txC-dpI-E
6. https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit