Politecnico Di Torino

ICT For Smart Societies

Final Lab Reports:
ICT for Health laboratories

Professor:
Monica Visintin, Ph.D.

Author:
Navid Yamini 235935

2017/ 2018

Table of Contents

1- Parkinson Dataset

1.1- Introduction

Parkinson's disease (PD) is a nervous system disorder that grows slowly step by step in a long term and affects movements. The cause of the PD is unknown, but it makes certain nerve cells (neurons) in the brain break down or die little by little. The severity of the PD is measured by neurologist during different sections and interviews. This process is time consuming and results can be different from one doctor to another. Because of that an automatic way is needed to estimate the severity of the PD faster and more accurately.

In the first laboratory, we were asked to work on Parkinson dataset. The objective of the lab was to implement MSE, Gradient algorithm, Steepest Descent algorithm, Ridge Regression, Principal Component Regression (PCR) and K-fold cross validation in order to apply them on the dataset to estimate and predict the severity of the Parkinson diseases among the patients and at the end compare the results of the said approaches with each other.

1.2- Dataset

The dataset chosen for this lab was Parkinson diseases dataset. The dataset is available on University of California Machine Learning Website[1]. It contains information related to 42 patients. Each patient was monitored for six months and the features that doctors were interested in were recorded automatically. Table 1- 1 shows those attributes.

Table 1- 1- Features' information of Parkinson dataset

ATTRIBUTE INFORMATION	
Attribute	**Description**
Subject#	Integer that uniquely identifies each subject
age	Subject age
sex	Subject gender '0' - male, '1' - female
test_ time	Time since recruitment into the trial. The integer part is the number of days since recruitment.
motor_ UPDRS	Clinician's motor UPDRS score, linearly interpolated
total_ UPDRS	Clinician's "total_ UPDRS" score, linearly interpolated
"Jitter (%)"	Several measures of variation in fundamental frequency
Jitter(Abs)	
Jitter: RAP	
Jitter: PPQ5	
Jitter: DDP	
Shimmer	Several measures of variation in amplitude
Shimmer (dB)	
Shimmer: APQ3	
Shimmer: APQ5	
Shimmer: APQ11	
Shimmer: DDA	
NHR	Two measures of ratio of noise to tonal components in the voice
HNR	

[1] https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring

RPDE	A nonlinear dynamical complexity measure
DFA	Signal fractal scaling exponent
PPE	A nonlinear measure of fundamental frequency variation

The feature that we were asked to forecast was "total_ UPDRS"[2]; therefore, this was our regressand or response variable.

1.3- Data Preparation

The original dataset has 5876 records, for each patient; time goes almost from 0 to 180 days, 5-6 times in subsequent blocks of data. In each 5-6 blocks new values have been recorded for each feature except for the total and motor UPDRS values since these values have been measured only once a day and copied for the rest records. For further information, it is essential to say that unfortunately some "test_time" values were submitted mistakenly as negative values.

For cleaning the data, we needed to create a new matrix with "test_times" going just from 0 to 180 for every person and to take the average of 5-6 blocks for the rest of the features, as well as taking the absolute values for negative "test_times". As a result, we had a matrix with 990 rows without negative values for the times. The next step was to divide the set into two subsets, one training set which included the first 36 patients and the other test set which included the rest of the patients. Furthermore, the datasets had to be normalized in a way that each feature would have a mean of zero and variance of 1. Please note that we needed to calculate the mean and variance of the train dataset to use it for normalizing both train and test sets.

1.4- Algorithms

The concept of linear regression is to find the relationship between the vector of independent variable x(n) which is a matrix with f columns corresponding to the number of features and n rows corresponding to the number of patients or records and dependent variable y(n) which is a vector with n rows. Equation 1-1 shows the linear regression model. In this equation the vector $\widehat{w}$.is a set of coefficients needed to be found and the ν is the measurement error.

$$y = Xw + \nu$$

Equation 1- 1- Linear Regression

Therefore, the idea of linear regression is to find the best weights that can predict y(n) by using x(n).

1-4-1- MSE

The idea behind minimum squared error is to estimate the $\widehat{w}$.in a way that minimizes the squared error (Equation 1-2).

$$e(w) = ||y - Xw||^2$$

Equation 1- 2- Squared Error

In this case to reach the minimum, we need to put the gradient of e(w) equal to zero. By doing that we will find the solution which is shown in Equation 1-3.

[2] Unified Parkinson's Disease Rating Scale

$$\widehat{w} = [X^T X]^{-1} X^T y$$

Equation 1- 3- estimated w

1-4-1-1- MSE Results

First, we need to set feature seven, "Jitter (%)", as a measured variable and plot the results. By plotting the results, we can realize that the algorithm is able to estimate the target value quite well in both training and test parts.

Table 1- 2- MSE algorithm errors for "Jitter (%)"

e(w) Train[3]	3.12530269694
e(w) Test	1.55133939686

As can be seen in Table 1- 2, the average error for test set is quite small and it shows that the MSE algorithm works quite well. We can see the results in Figures 1-1 to 1-7. These results are related to predicting "Jitter (%)". As it is noticeable in the line plot Figure 1-1 and scatter plot Figure 1-5, the real "y" values and estimated "y" values are almost close to each other in predicting training set. For the test on Figure 1-2 and Figure 1-6 it is noticeable that the algorithm underestimates the "y" value a little bit but in general it works well. Figures 1-3 and 1-4 are histogram plots for training set and test set. They show the differences between the real values and the predicted ones. These figures show that the error values are almost normally distributed. Finally, Figure 1-7 shows the weighted values; higher values are assigned to features number 8 and 11 which are Shimmer: APQ3 and Shimmer: DDA (please refer to table 1-3 to see the Features' indexes). In conclusion, we can say that these features play a more important role in predicting the "Jitter (%)" value.

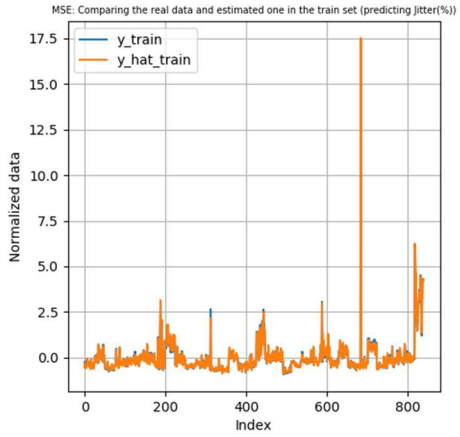


Figure 1- 1- MSE, Comparing the real data and estimated one in the train set predicting "Jitter (%)"

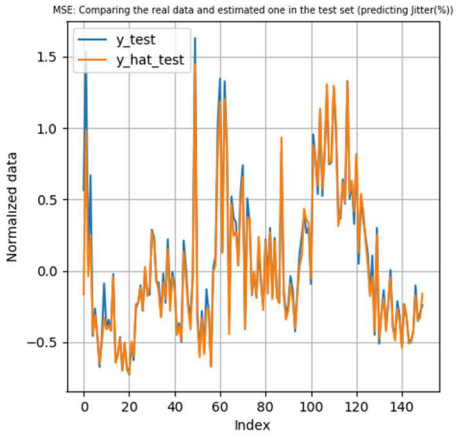


Figure 1- 2- MSE, Comparing the real data and estimated one in the test set predicting "Jitter (%)"

[3] e(w) refers to Squared Error

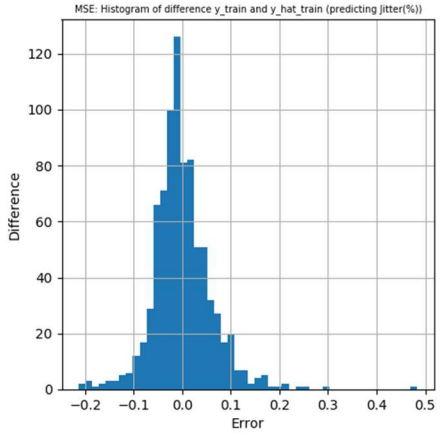


Figure 1- 3- MSE error histogram for train set predicting "Jitter (%)"

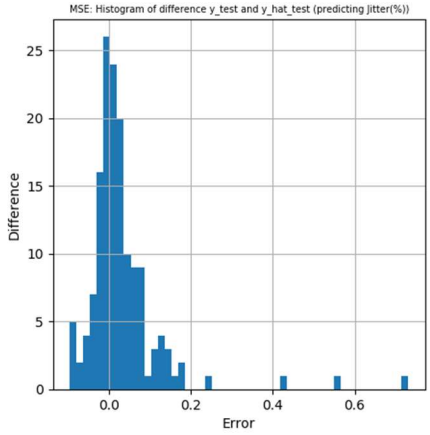


Figure 1- 4- MSE error histogram for test set predicting "Jitter (%)"

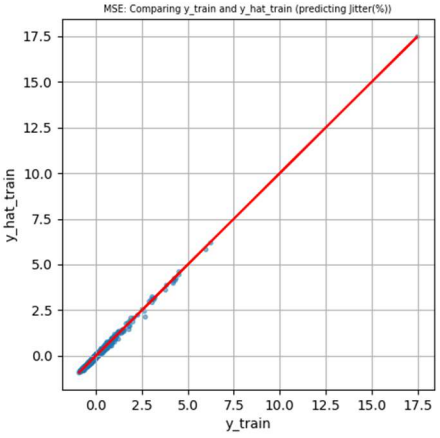


Figure 1- 5- MSE scatter plot for train set predicting "Jitter (%)"

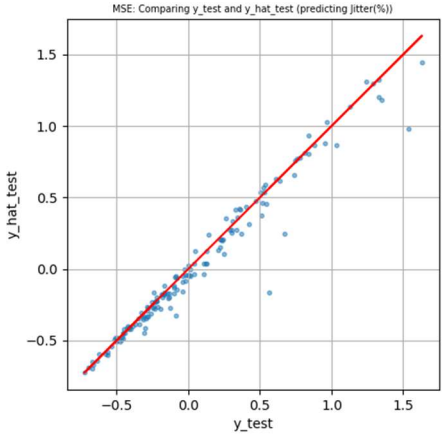


Figure 1- 6- MSE scatter plot for test set predicting "Jitter (%)"

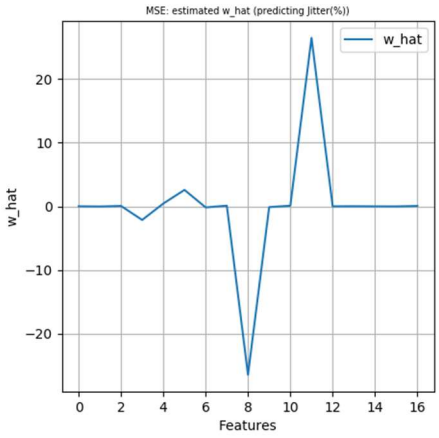


Figure 1- 7- MSE estimated w for predicting "Jitter (%)"

Table 1- 3- Features' Index for predicting "Jitter (%)"

Features	Index
motor_ UPDRS	0
total_ UPDRS	1
Jitter(Abs)	2
Jitter: RAP	3
Jitter: PPQ5	4
Jitter: DDP	5
Shimmer	6
Shimmer (dB)	7
Shimmer: APQ3	8
Shimmer: APQ5	9
Shimmer: APQ11	10
Shimmer: DDA	11
NHR	12
HNR	13
RPDE	14
DFA	15
PPE	16

Now the next step is to use MSE algorithm to predict "total_ UPDRS". We need to replace out "y" set with "total_ UPDRS" values and repeat the same steps that we did for "Jitter (%)".

Table 1- 4- MSE algorithm errors for "total_ UPDRS"

e(w) Train	86.5694294829
e(w) Test	5.72038859589

As you can see in Table 1- 4, this time error values are higher than the previous time, and this result shows that "Jitter (%)" is more related to other features than "total_ UPDRS".

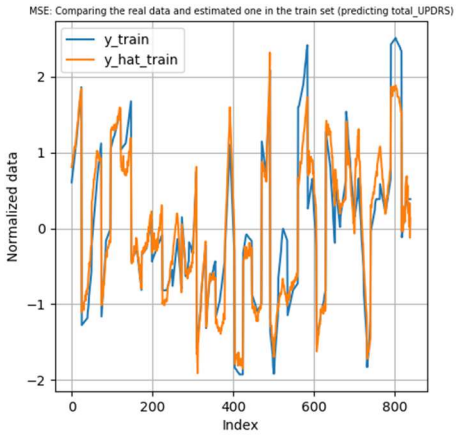


Figure 1- 8- MSE, Comparing the real data and estimated one in the train set predicting "total_ UPDRS"

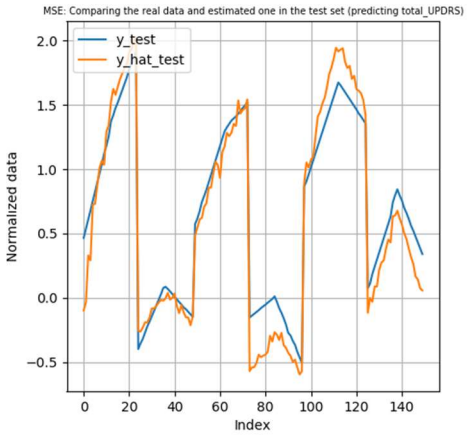


Figure 1- 9- MSE, Comparing the real data and estimated one in the test set predicting "total_ UPDRS"

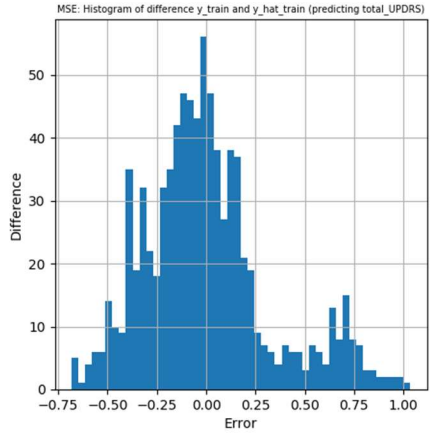


Figure 1- 10- MSE error histogram for train set predicting "total_ UPDRS"

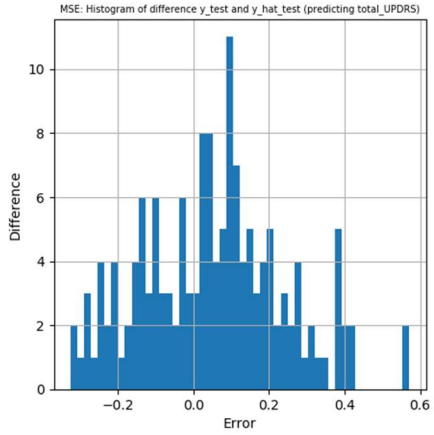


Figure 1- 11- MSE error histogram for test set predicting "total_ UPDRS"

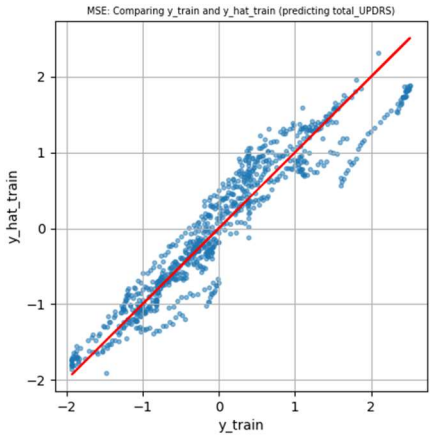


Figure 1- 12- MSE scatter plot for train set predicting "total_ UPDRS"

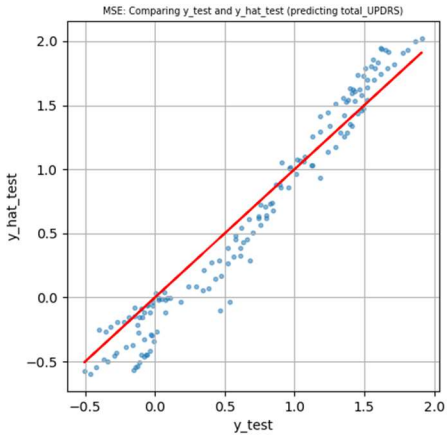


Figure 1- 13- MSE scatter plot for test set predicting "total_ UPDRS"

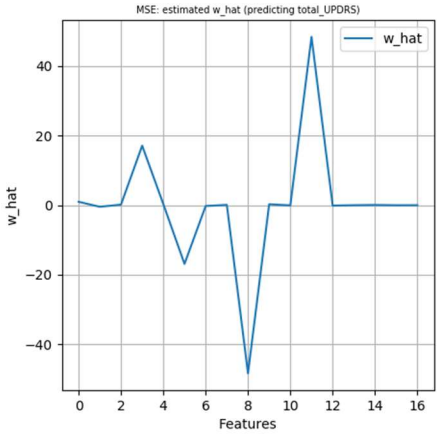


Figure 1- 14- MSE estimated w for predicting "total_ UPDRS"

Figures 1-8 to 1-14 show the results for estimating "total_ UPDRS" by using MSE algorithm. From Figures 1-8 and 1-12 it is understood that in a train set, for most of the values, the algorithm overestimates the

values; on the other hand, Figures 1-9 and 1-13 show that we have overestimated values for higher ones and underestimated values for the lower ones in the test set. Figure 1-10 shows that the error histogram for train set has a positively skewed distribution due to the negative average but on the other hand, $\hat{y}$ values related to the test have a left skewed distribution since we have a positive average for it (shown in Figure 1-11). Finally, Figure 1-14 shows that this time four features have a different weight than zero. They are features 3,5,8 and 11 (please refer to table 1-5 to see the Features' indexes).

Table 1- 5- Features' Index for predicting "total_ UPDRS"

Features	Index
motor_ UPDRS	0
"Jitter (%)"	1
Jitter(Abs)	2
Jitter: RAP	3
Jitter: PPQ5	4
Jitter: DDP	5
Shimmer	6
Shimmer (dB)	7
Shimmer: APQ3	8
Shimmer: APQ5	9
Shimmer: APQ11	10
Shimmer: DDA	11
NHR	12
HNR	13
RPDE	14
DFA	15
PPE	16

1-4-2- Iterative, Gradient Algorithm

Inverting the matrix is a complex procedure. To avoid doing that, iterative, gradient algorithm is implemented. In this algorithm we used an iterative approach to estimate the w. In the first step, we needed to start with an initial guess for $\hat{w}$ values that could be random variables and we called it $\hat{w}$ (i)[4]. Next, we needed to calculate the gradient and update the $\hat{w}$ (i+1) value by using the computed gradient and γ[5] which is summarized in Equations 1-4 and 1-5. The value of $\hat{w}$.will be updated in a loop until reaching to the stopping condition[6] which is shown in Equation 1-6.

$$\nabla e(\hat{w}(i)) = -2X^T y \quad + 2X^T X\, \hat{w}(i)$$

Equation 1- 4- Gradient

$$\hat{w}(i+1) = \hat{w}(i) - \gamma \nabla e(\hat{w}(i))$$

Equation 1- 5- update the estimated w

$$||\hat{w}(i+1) - \hat{w}(i)|| \quad < \varepsilon$$

Equation 1- 6- Stopping Condition

The learning coefficient γ can be seen as a jumping step towards the optimum solution. Setting a right value for it is a critical step, because setting γ too large makes the algorithm take large steps and jump

[4] i starts from 0 to n

[5] Learning Coefficient $\gamma > 0$

[6] ε can be set

around the optimal solution instead of reaching it. On the other hand, setting γ too small will take a lot of time and iterations for the algorithm to reach the solution. For this experiment, $\varepsilon = 1 \times 10^{-7}$ and $\gamma = 1 \times 10^{-6}$. Both values were chosen by trial and error approach.

1-4-2-1- Iterative, Gradient Algorithm Results

Like the previous time, first we estimated the "Jitter (%)" and after that we predicted the "total_ UPDRS". Table 1-6 shows the results error. The errors are almost like the values that we reached in MSE.

Table 1- 6: Iterative, Gradient algorithm errors for "Jitter (%)"

e(w) Train	3.13421560435
e(w) Test	1.54155959962

These results were obtained after 430873 iterations.

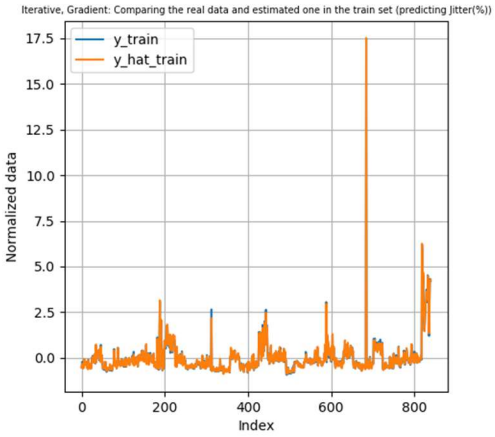


Figure 1- 15- Iterative, Gradient: Comparing the real data and estimated one in the train set predicting "Jitter (%)"

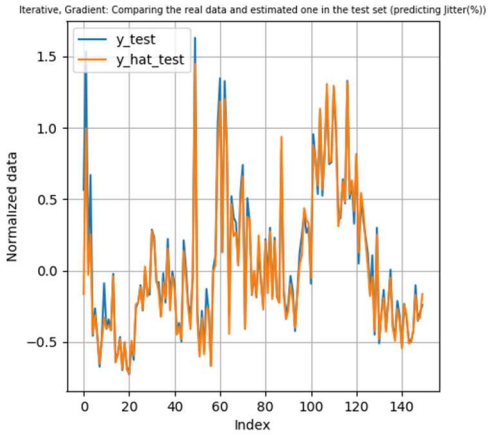


Figure 1- 16- Iterative, Gradient: Comparing the real data and estimated one in the test set predicting "Jitter (%)"

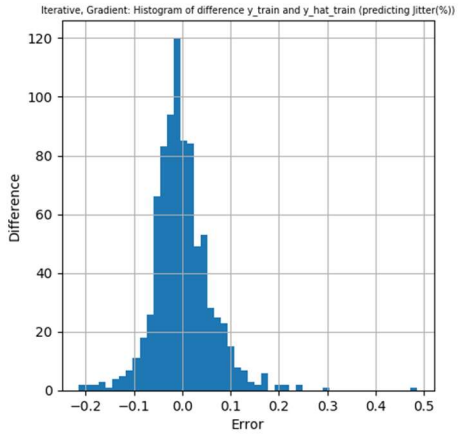


Figure 1- 17- Iterative, Gradient: error histogram for train set predicting "Jitter (%)"

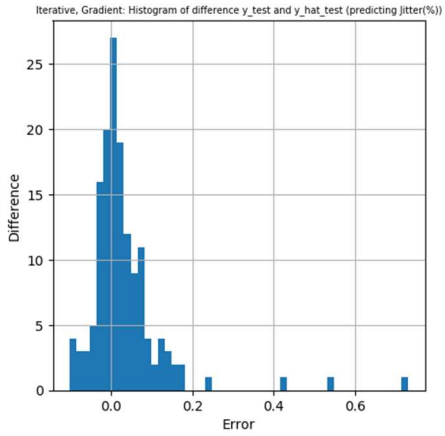


Figure 1- 18- Iterative, Gradient: error histogram for test set predicting "Jitter (%)"

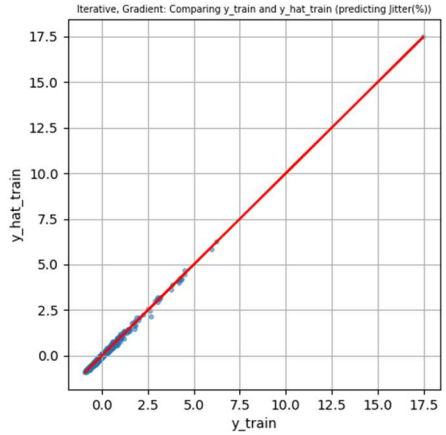


Figure 1- 19- Iterative, Gradient: scatter plot for train set predicting "Jitter (%)"

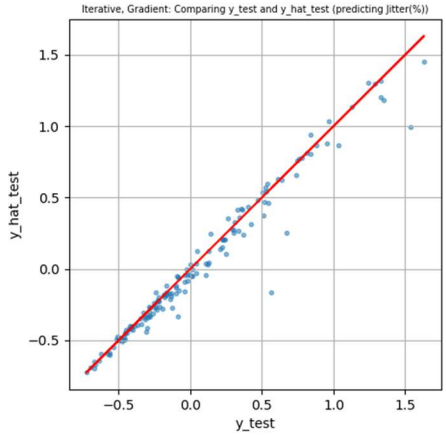


Figure 1- 20- Iterative, Gradient: scatter plot for test set predicting "Jitter (%)"

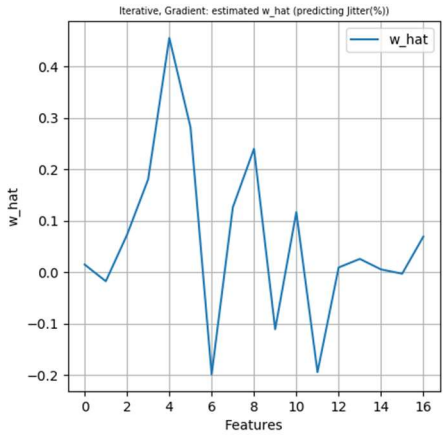


Figure 1- 21- Iterative, Gradient: estimated w for predicting "Jitter (%)"

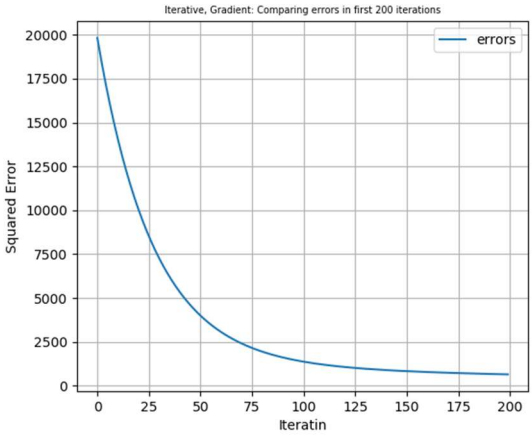


Figure 1- 22- Iterative, Gradient: The Squared error for predicting "Jitter (%)"

As it is noticeable in Figures 1- 15 to 1- 20, the outputs are almost like the MSE solution's results. Nevertheless, this time as is shown in Figure 1-21, the estimated $\widehat{w}$.is within [-0.2, 0.4] range, which is smaller than the previous time. Furthermore, this time, higher values are assigned to features number 4, 6, 9, 10 and 11 which are Jitter: PPQ5, Shimmer, Shimmer: APQ5, Shimmer: APQ11 and Shimmer: DDA (please refer to Table 1-3 to see the Features' indexes). In addition, as we are using the iterative mode, each time the difference between real values and estimated values is smaller than the previous iteration. Figure 1- 22 shows the squared error for the 200 first iterations which decreases exponentially.

Now it is time to use this solution to predict "total_ UPDRS".

Table 1- 7- Iterative, Gradient algorithm errors for "total_ UPDRS"

e(w) Train	86.6284721287
e(w) Test	5.64387005614

These results were obtained after 692767 iterations. Therefore, reaching optimal solution for estimating "total_ UPDRS" takes more iterations and time in comparison with "Jitter (%)". Table 1-7 shows the results error. The errors are almost like the values that we reached in MSE.

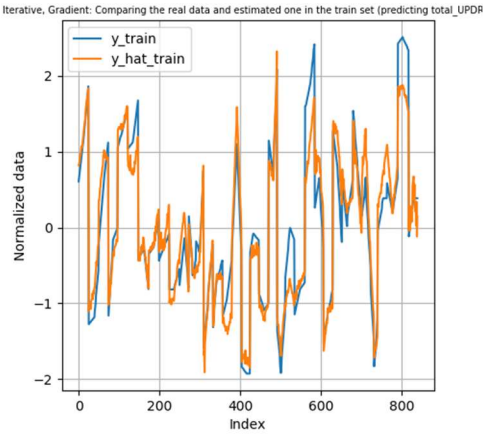


Figure 1- 23- Iterative, Gradient: Comparing the real data and estimated one in the train set predicting "total_ UPDRS"

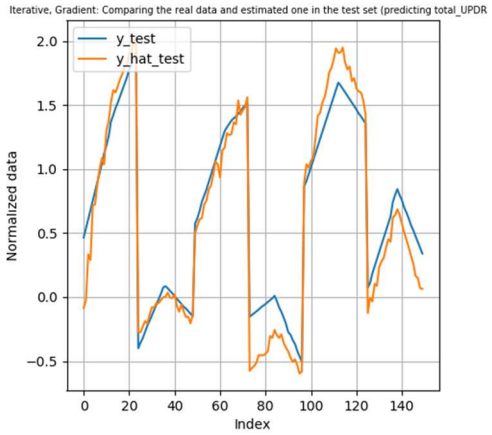


Figure 1- 24- Iterative, Gradient: Comparing the real data and estimated one in the test set predicting "total_ UPDRS"

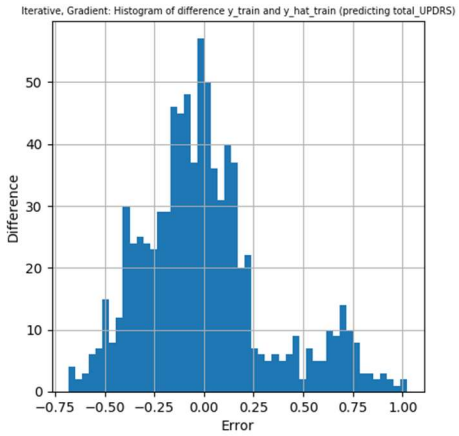


Figure 1- 25- Iterative, Gradient: error histogram for train set predicting "total_ UPDRS"

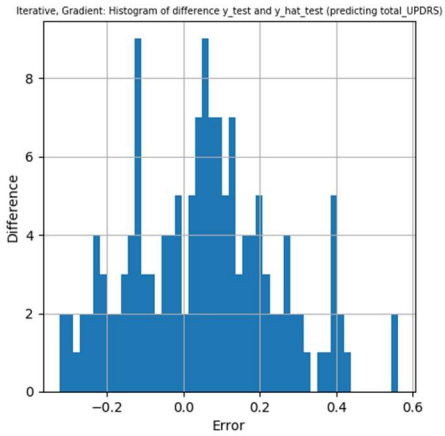


Figure 1- 26- Iterative, Gradient: error histogram for test set predicting "total_ UPDRS"

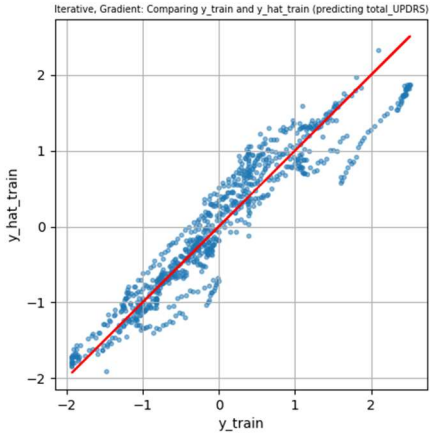


Figure 1- 27- Iterative, Gradient: scatter plot for train set predicting "total_ UPDRS"

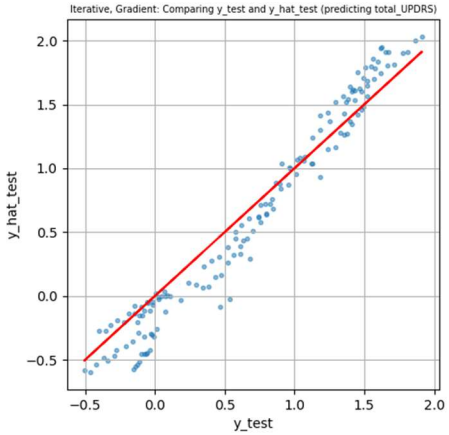


Figure 1- 28- Iterative, Gradient: scatter plot for test set predicting "total_ UPDRS"

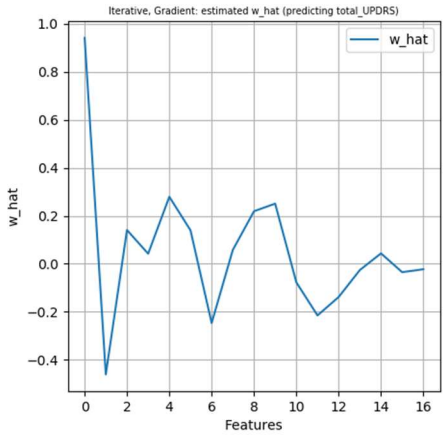


Figure 1- 29- Iterative, Gradient: estimated w for predicting "total_ UPDRS"

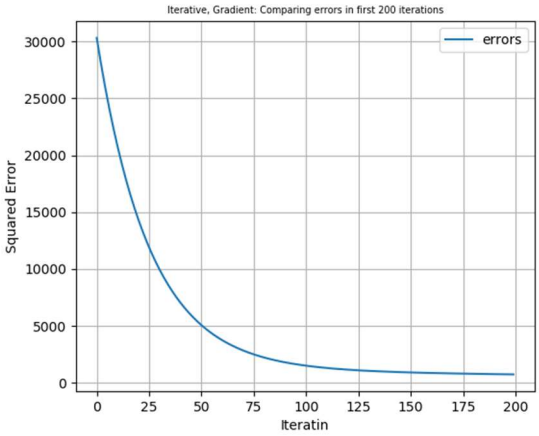


Figure 1- 30- Iterative, Gradient: The Squared error for predicting "total_ UPDRS"

Comparing the obtained results with the results that we saw in part 1-4-1-1, we observe that the results are almost the same. We can see that in Figures 1-23, 1-27. Again, we have underestimation and scatter plots in Figures 1-24 and 1-29 show that we have underestimation for lower values and overestimation for higher "y" values in test set. Furthermore, Figure 1-25 shows that the error histogram for train set has a positively skewed distribution due to the negative average; on the other hand, in Figure 1-26 we have a double-peaked or bimodal distribution. By comparing this result with estimating "Jitter (%)" by using iterative, gradient algorithm, as we expected we obtained better results for "Jitter (%)" estimation. Also, we have completely different results for $\widehat{w}$ (Figure 29). The higher values belong to features 0, 1, 4, 6, 8 and 11(please refer to Table 1-5 to see the Features' indexes). Figure 1-30 shows that the first calculated squared error is much higher than the previous time. At the end, by comparing the execution time of MSE algorithm with Iterative, Gradient we can conclude that MSE works faster.

1-4-3- The Steepest Descent Algorithm

The Steepest Descent algorithm, like the previous algorithm, works in the iterative way. The difference between this algorithm and Iterative Gradient is in finding the optimum value for γ and is updated in each iteration, so we expected that the Steepest Descent algorithm works faster than Iterative Gradient.

Therefore, the formula to calculate the value of $\hat{w}$ (i+1) need to be changed in the following way (Equation 1-7).

$$\hat{w}(i+1) = \hat{w}(i) - \frac{||\nabla e(\hat{w}(i))||^2}{\nabla e(\hat{w}(i))^T \mathrm{H}\big(\hat{w}(i)\big)\nabla e(\hat{w}(i))} \nabla e(\hat{w}(i))$$

Equation 1- 7- update the estimated w

The H matrix shown in Equation 1-7 is a Hessian matrix. The formula needed to calculate it is shown in Equation 1-8.

$$\mathrm{H}(\hat{w}(i)) = 4X^T X$$

Equation 1- 8- Hessian Matrix

Please note that we used the same stopping condition as the previous time (Equation 1-6).

1-4-3-1- The Steepest Descent Algorithm Results

First, we predicted "Jitter (%)" with errors shown in Table 1-8. Comparing the errors with what we saw in the Iterative Gradient algorithm, we see almost the same values.

Table 1- 8- The Steepest Descent algorithm errors for "Jitter (%)"

e(w) Train	3.13198110333
e(w) Test	1.56016784377

Figures 1-31 to 1-38 show the Steepest Descent algorithm output for predicting "Jitter (%)". It is obvious that results are as Iterative Gradient that we saw.
Since the learning coefficient has been updated in every iteration and set to the optimum, we expected to obtain the results faster than before. This time, the results were obtained after 929 iterations which means that this algorithm works much faster than Iterative Gradient. Furthermore, as shown in Figure 1-38, the squared error decremented faster. These results demonstrate our hypothesis. In general, it is possible to say that Steepest Descent takes less time to obtain the results, but it is more complex to implement. Figure 1-37 shows the calculated weights. This time features 4, 5, 6, 7,8, 9, 10, 11 have higher values (please refer to Table 1-3 to see the Features' indexes).

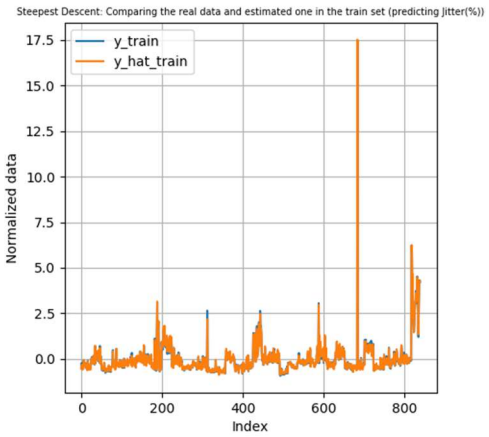


Figure 1- 31- Steepest Descent: Comparing the real data and estimated one in the train set predicting "Jitter (%)"

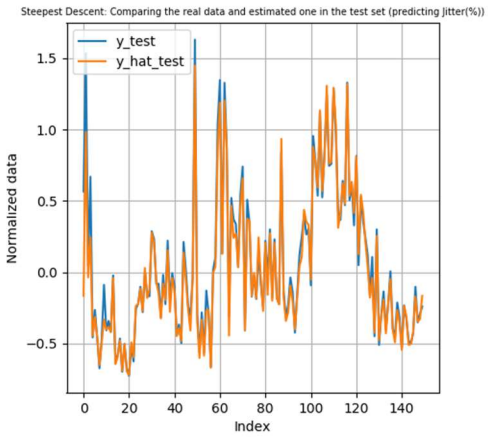


Figure 1- 32- Steepest Descent: Comparing the real data and estimated one in the test set predicting "Jitter (%)"

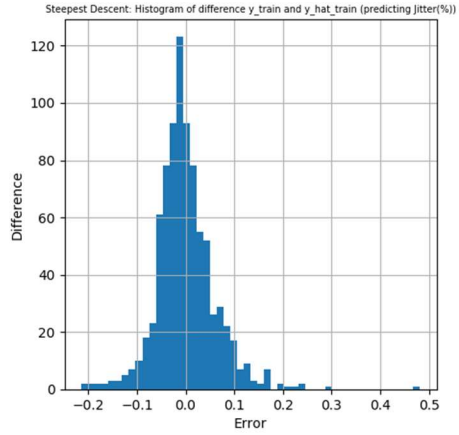


Figure 1- 33- Steepest Descent: error histogram for train set predicting "Jitter (%)"

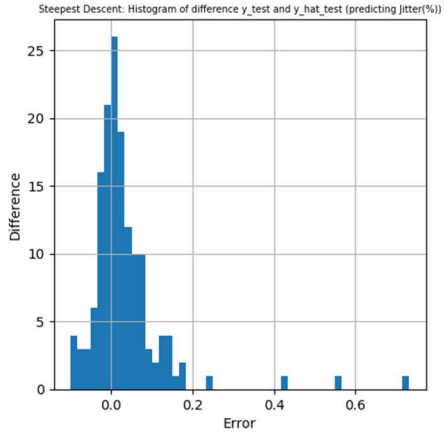


Figure 1- 34- Steepest Descent: error histogram for test set predicting "Jitter (%)"

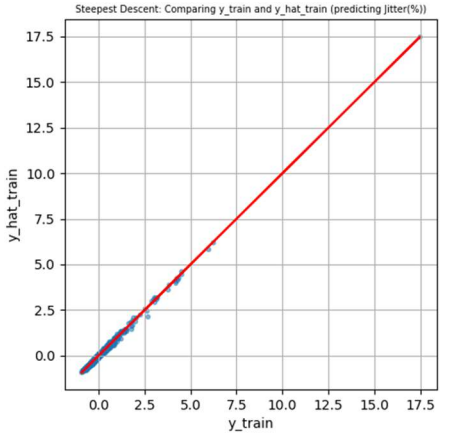


Figure 1- 35- Steepest Descent: scatter plot for train set predicting "Jitter (%)"

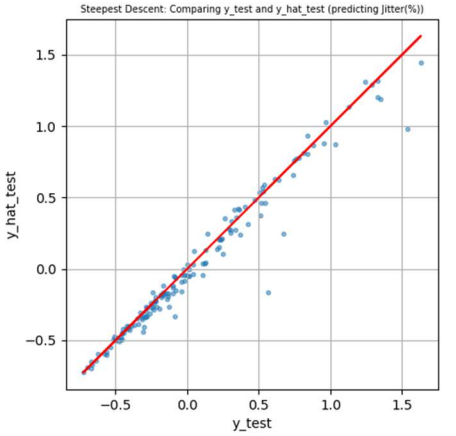


Figure 1- 36- Steepest Descent: scatter plot for test set predicting "Jitter (%)"

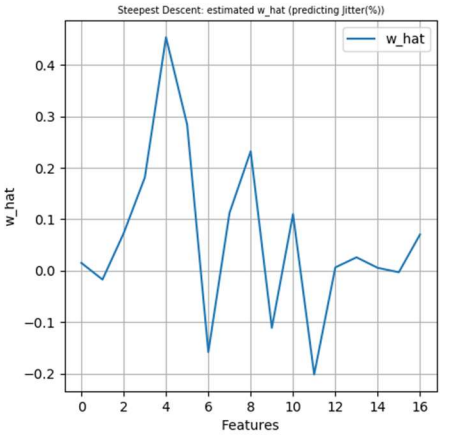
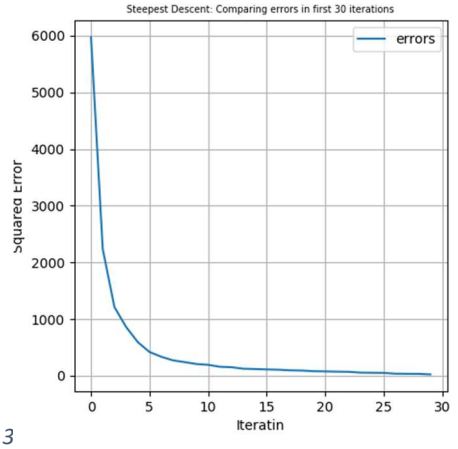


Figure 1- 37- Steepest Descent: estimated w for predicting "Jitter (%)"



3

Figure 1- 38- Steepest Descent: The Squared error for predicting "Jitter (%)"

Like other algorithms in second steep, the "total_ UPDRS" has been set as a dependent value. Table 1-9 shows the error results.

Table 1- 9- Steepest Descents algorithm errors for "total_ UPDRS"

e(w) Train	86.5920951092
e(w) Test	5.72748963556

These results were obtained after 34065 iterations and the squared error decremented faster (Figure 1-46). This proves that Steepest Decants works faster than Iterative Gradient. Therefore, reaching the optimal solution for estimating "total_ UPDRS" takes more iterations and time in comparison with "Jitter (%)". Figures 1-39 to 1-46 show the Steepest Descent algorithm output for predicting "total_ UPDRS". The results are almost like previous times except by comparing Figure 1-41 with Figure 1-42 it is noticeable that error values in training set are distributed better. Furthermore, Figure 1-45 shows that this time features 3, 5, 8 and 11 weighted more than other features (please refer to Table 1-5 to see the Features' indexes).

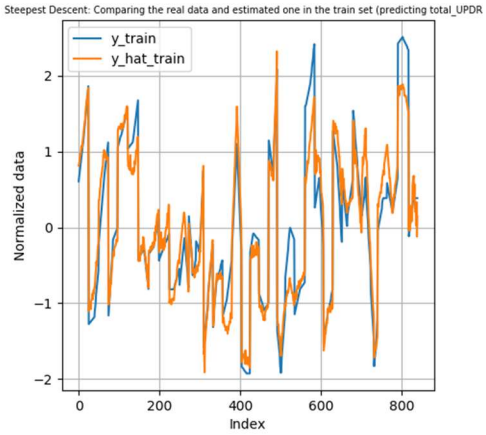


Figure 1- 39- Steepest Descent: Comparing the real data and estimated one in the train set predicting "total_ UPDRS"

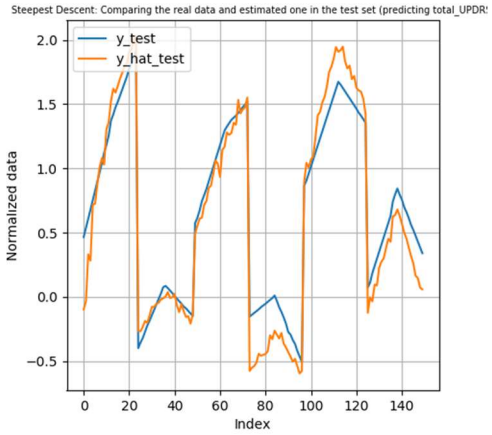


Figure 1- 40- Steepest Descent: Comparing the real data and estimated one in the test set predicting "total_ UPDRS"

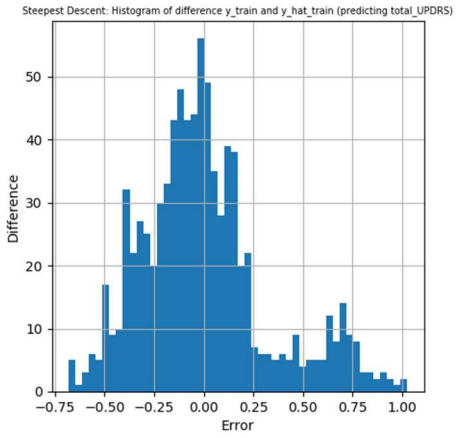


Figure 1- 41- Steepest Descent: error histogram for train set predicting "total_ UPDRS"

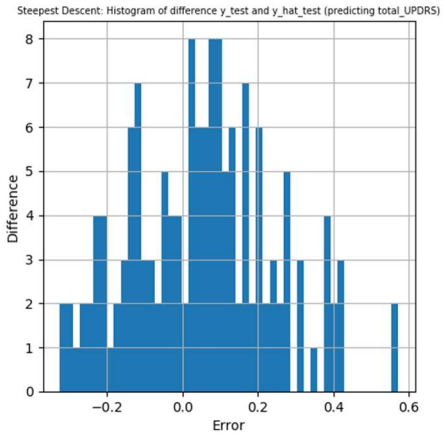


Figure 1- 42- Steepest Descent: error histogram for test set predicting "total_ UPDRS"

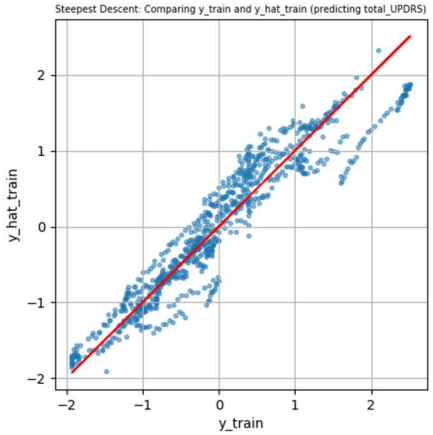


Figure 1- 43- Steepest Descent: scatter plot for train set predicting "total_ UPDRS"

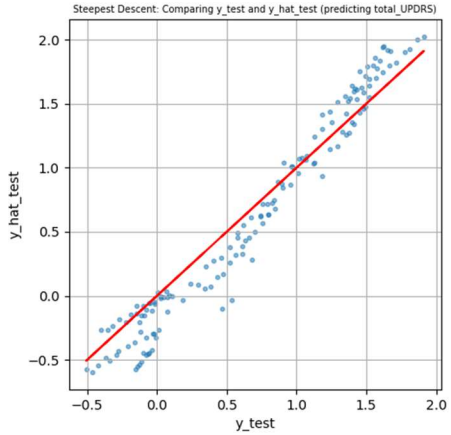


Figure 1- 44- Steepest Descent: scatter plot for test set predicting "total_ UPDRS"

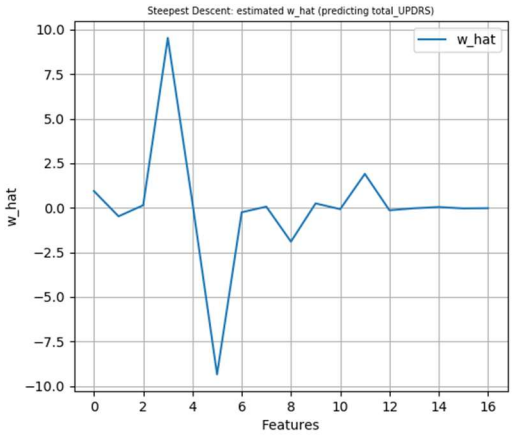


Figure 1- 45- Steepest Descent: estimated w for predicting "total_ UPDRS"

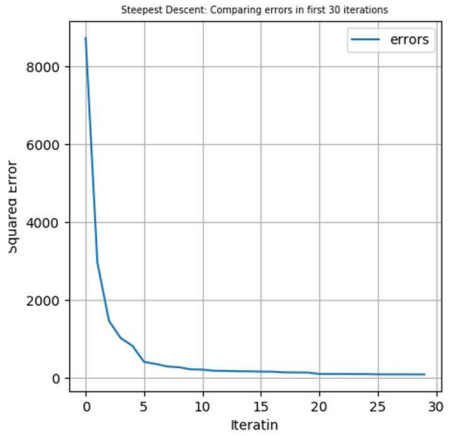


Figure 1- 46- Steepest Descent: The Squared error for predicting "total_ UPDRS"

1-4-4- The Ridge Regression

The idea behind ridge regression is to penalize vector $\hat{w}$. When there is a large value of error or noise in the $y(n) = Xw + \nu$, it is possible that vector $\hat{w}$ gets large and strange values. In this case, we will have new problems that can be solved by Equation 1-9.

$$min\ ||y - Xw||^2 + \lambda||w||^2$$

Equation 1- 9

Here, the $\hat{w}$. instead of being a set of parameters is a random vector with i.i.d[7] Gaussian variable with zero mean and variance equal to s^2. By setting the gradient of the objective function equal to zero, we can find the value of $\hat{w}$.(shown in Equation 1-10)

$$\hat{w} = (X^T X + \lambda I]^{-1} X^T y_{\text{meas}}$$

Equation 1- 10- Ridge Regression

[7] Independent and Identically Distributed

λ is a Lagrange multiplier and different values of λ will lead to different degrees of overfitting. Which our experiment set to 0.1 by trial and error.

1-4-4-1- The Ridge Regression Algorithm Results

Like the previous times, we started to predict "Jitter (%)" first. Errors are shown in Table 1-10 which are almost the same as what we saw before.

Table 1- 10- The Ridge Regression algorithm errors for "Jitter (%)"

e(w) Train	3.1323729523
e(w) Test	1.55687889105

Figures 1-47 to 1-53 show the Ridge Regression algorithm output. It is obvious that the results are similar to the previous algorithms and it shows that the algorithm works well. Figure 1-53 shows the calculated weights; this time features 4, 5, 6, 7, 9, 10 have higher values (please refer to Table 1-3 to see the Features' indexes).

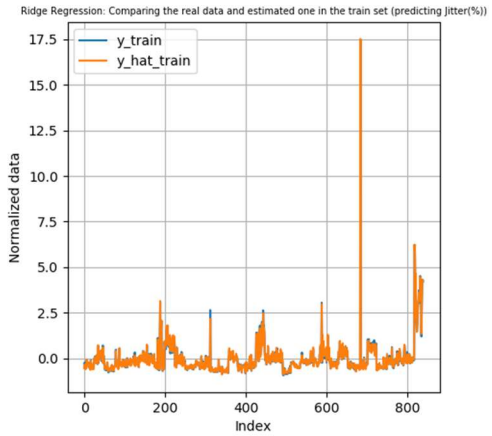


Figure 1- 47- Ridge Regression: Comparing the real data and estimated one in the train set predicting "Jitter (%)"

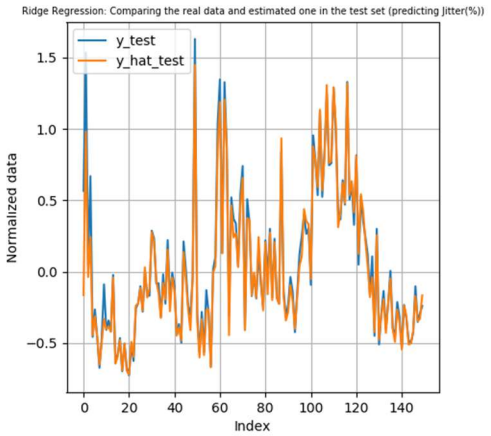


Figure 1- 48- Ridge Regression: Comparing the real data and estimated one in the test set predicting "Jitter (%)"

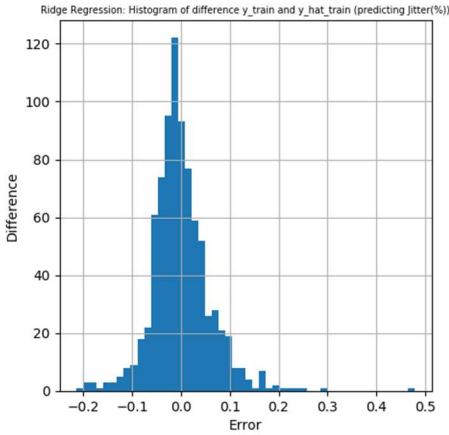


Figure 1- 49- Ridge Regression: error histogram for train set predicting "Jitter (%)"

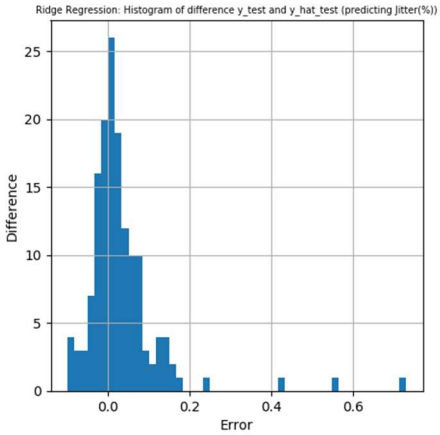


Figure 1- 50- Ridge Regression: error histogram for test set predicting "Jitter (%)"

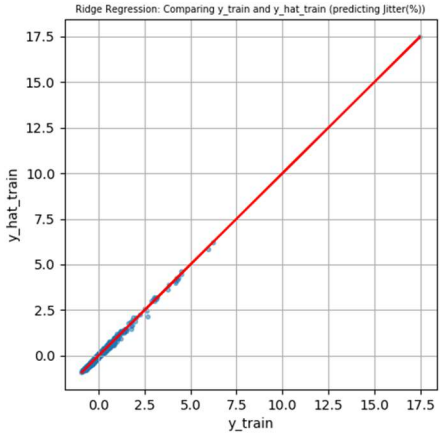


Figure 1- 51- Ridge Regression: scatter plot for train set predicting "Jitter (%)"

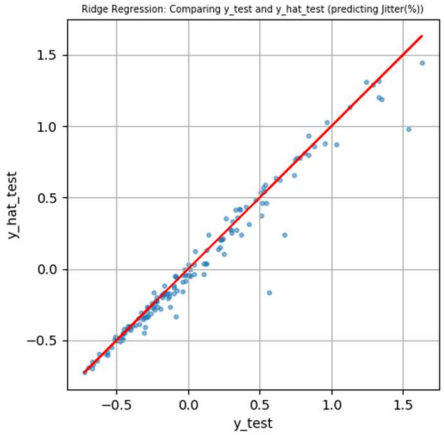


Figure 1- 52- Ridge Regression: scatter plot for test set predicting "Jitter (%)"

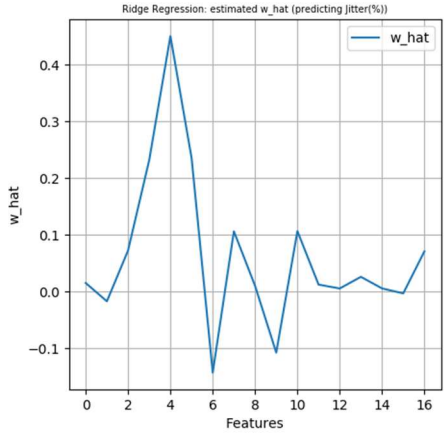


Figure 1- 53- Ridge Regression: estimated w for predicting "Jitter (%)"

Now for the next step, we used this algorithm to predict "total_ UPDRS". Table 1-11 shows the error results.

Table 1- 11- Ridge regression algorithm errors for "total_ UPDRS"

e(w) Train	86.6289910412
e(w) Test	5.64742041221

The results are shown in Figures 1-54 to 1-60. Comparing the results with the previous one we can observe that the results are the same as other algorithms. The scatter plot in Figure 1-57 shows that for the test set the algorithm overestimated the higher "y" values and underestimated the low values. In the histogram plot in Figure 1-56, we see a positively skewed distribution due to the negative average and the error values for training sets are distributed more equally comparing with the test set in Figure 1-57. Furthermore, in Figure 1-58 we can see that the error diagram for test set has two peaks

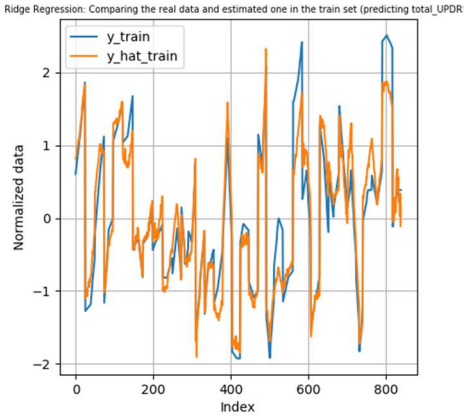


Figure 1- 54- Ridge regression: Comparing the real data and estimated one in the train set predicting "total_ UPDRS"

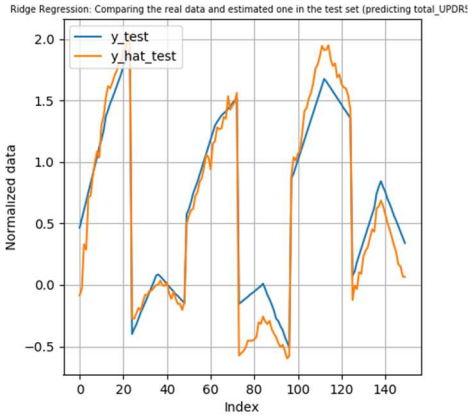


Figure 1- 55- Ridge regression: Comparing the real data and estimated one in the test set predicting "total_ UPDRS"

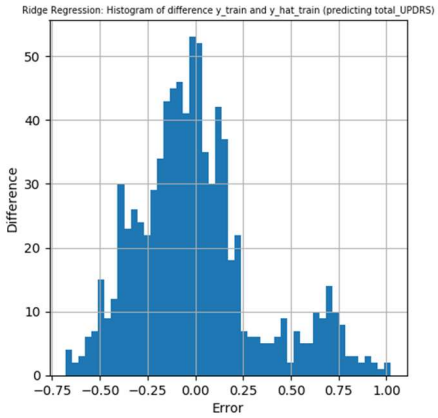


Figure 1- 56- Ridge regression: error histogram for train set predicting "total_ UPDRS"

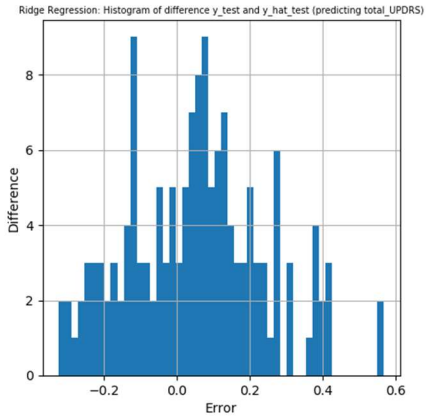


Figure 1- 57- Ridge regression: error histogram for test set predicting "total_ UPDRS"

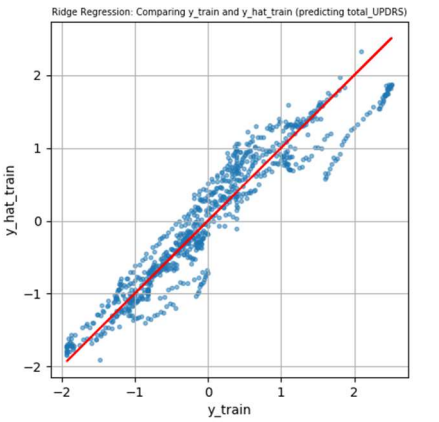


Figure 1- 58- Ridge regression: scatter plot for train set predicting "total_ UPDRS"

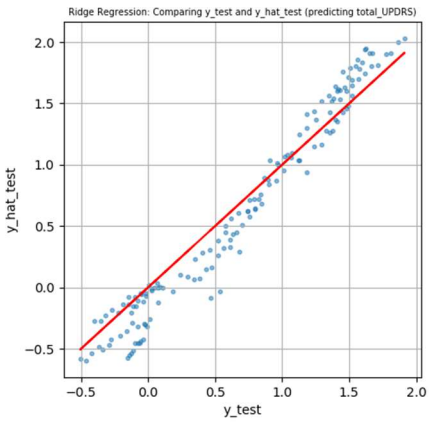


Figure 1- 59- Ridge regression: scatter plot for test set predicting "total_ UPDRS"

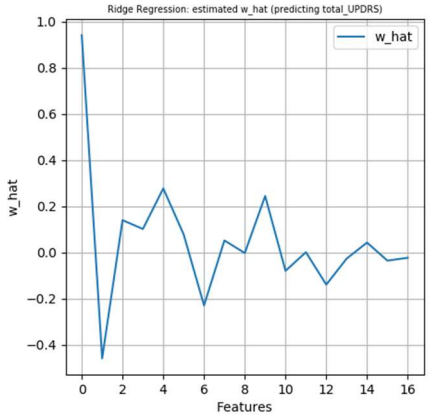


Figure 1- 60- Ridge regression: estimated w for predicting "total_ UPDRS"

1-4-5- Principal Component Regression (PCR)

The fifth algorithm implemented on Parkinson dataset was Principal Component Regression (PCR). The idea behind PCR is, instead of using all the features to estimate the measured variable, to predict response variables by using only L numbers of them which are uncorrelated. Using this approach will cause to decrease the dimension of the problem. Decreasing the value of L will lead to a larger mean value of error. Equation 1-11 shows the formula for calculating $\hat{y}$.

$$\hat{y}^{(L)}(n) = x(n)w^{(L)}$$

Equation 1- 11- Regression by using L features

$w^{(L)}$ can be calculated by using Equation 1-12.

$$w^{(L)} = \frac{1}{N} U^{(L)} \left(\Lambda^{(L)}\right)^{-1} \left(U^{(L)}\right)^{T} X^{T} y \ \in \ \mathbb{R}^{F\times 1}$$

Equation 1- 12- Calculating $w^{(L)}$

In Equation 1-12, $\Lambda^{(L)}$ is the diagonal matrix with the largest L eigenvalues and $U^{(L)}$ is the matrix with the L eigenvectors corresponding to the largest L eigenvalues.

1-4-5-1- The PCR Results

As mentioned before, the PCR algorithm works with a subset of features to estimate the coefficients. To better understand the way the PCR algorithm works, the algorithm was ran over the dataset four different times with different numbers of selected features. It was in fact predictable that reducing the number of features would increase the mean value of error. In the first case, the PCR was executed to predict "Jitter (%)". The error results are shown in Table 1-12.

Table 1- 12- PCR errors for "Jitter (%)"

	PCR1	PCR2	PCR3	PCR4
e(w) Train	3.13094357442	3.60791174775	4.21127216569	6.72975389889
e(w) Test	1.55798739094	1.06803365897	1.16374072855	1.66083809043
Percentage	100%	99.9%	99%	97%
No. Selected Features	17	11	8	6

Figures 1-61 to 1-68 show the output for PCR algorithm. Please note that they show only the results related to PCR2.

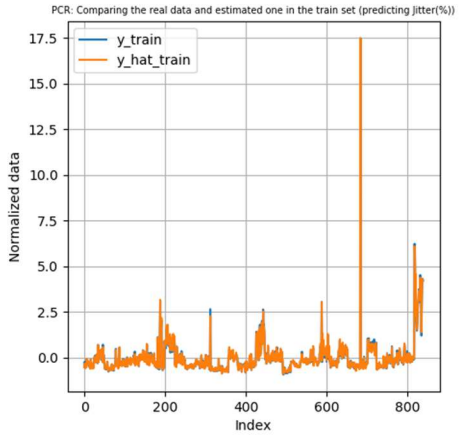


Figure 1- 61- PCR: Comparing the real data and estimated one in the train set predicting "Jitter (%)"

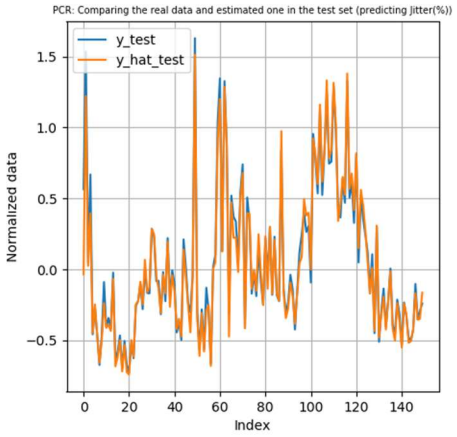


Figure 1- 62- PCR: Comparing the real data and estimated one in the test set predicting "Jitter (%)"

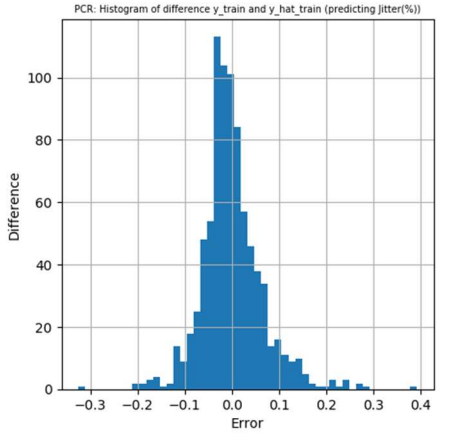


Figure 1- 63- PCR: error histogram for train set predicting "Jitter (%)"

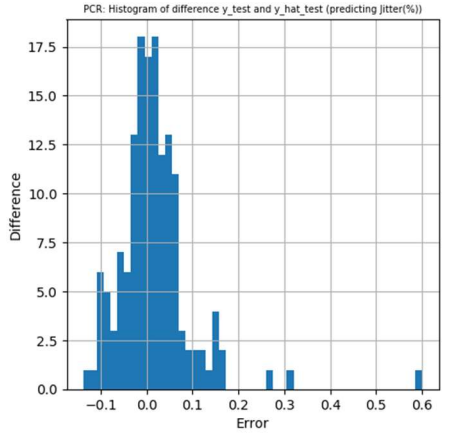


Figure 1- 64- PCR: error histogram for test set predicting "Jitter (%)"

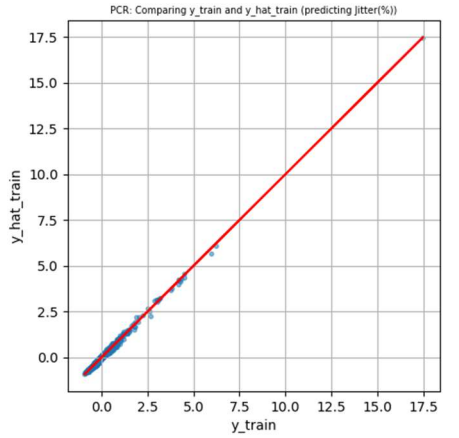


Figure 1- 65- PCR: scatter plot for train set predicting "Jitter (%)"

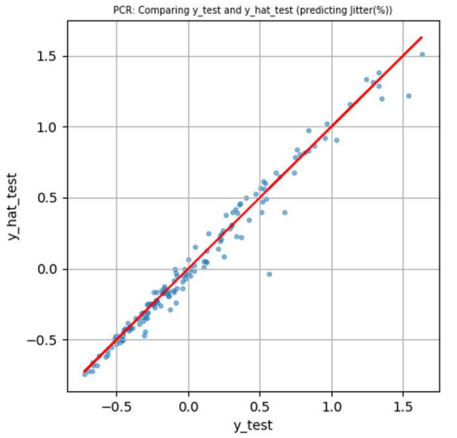


Figure 1- 66- PCR: scatter plot for test set predicting "Jitter (%)"

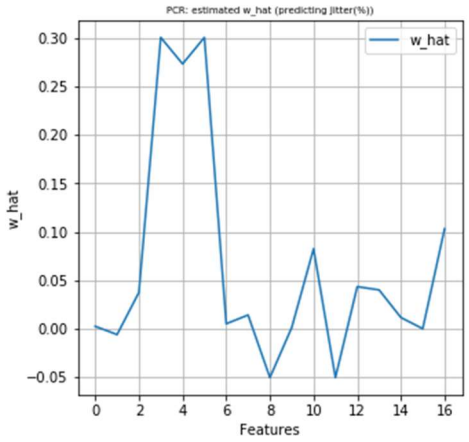


Figure 1- 67- PCR: estimated w for predicting "Jitter (%)"

By comparing histogram plots (Figure 1-63 and Figure 1-64) with the previous one we can see that the error range especially for the test set is higher than before. Furthermore, Figure 1-67 shows that features 3,4,5 are most important for predicting "y" values. (Please refer to Table 1-3 to see the Features' indexes).

For the second step, we used the PCR to predict "total_ UPDRS". Table 11 shows the error results.

Table 1- 13- PCR errors for "total_ UPDRS"

	PCR1	PCR2	PCR3	PCR4
e(w) Train	86.5883626311	88.5546339328	92.7578782479	136.531949584
e(w) Test	5.75817652564	5.62172660779	7.2774563524	8.61190465552
Percentage	100%	99.9%	99%	97%
No. Selected Features	17	10	8	5

Figures 1-68 to 1-74 show the output for PCR algorithm predicting "total_ UPDRS". Please note that they are depicting only the results related to PCR2.

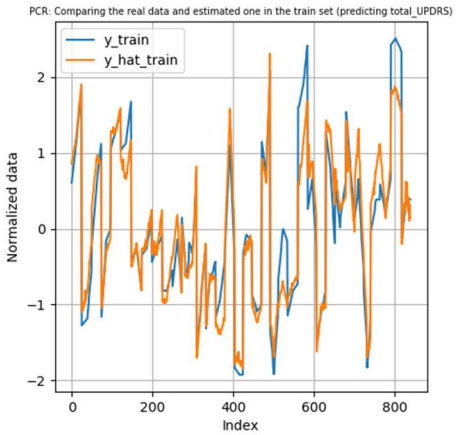


Figure 1- 68- PCR: Comparing the real data and estimated one in the train set predicting "total_ UPDRS"

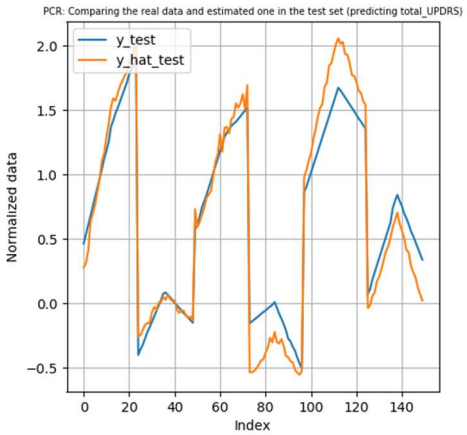


Figure 1- 69- PCR: Comparing the real data and estimated one in the test set predicting "total_ UPDRS"

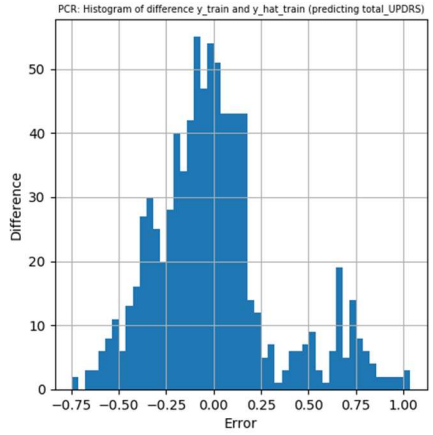


Figure 1- 70- PCR: error histogram for train set predicting "total_ UPDRS"

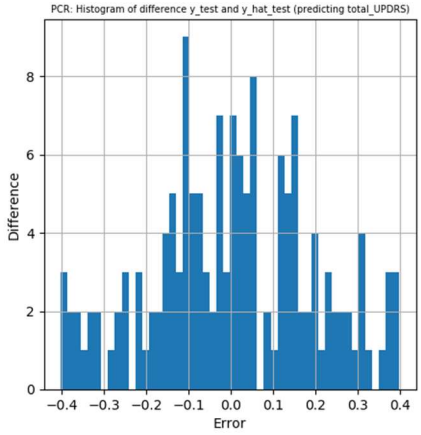


Figure 1- 71- PCR: error histogram for test set predicting "total_ UPDRS"

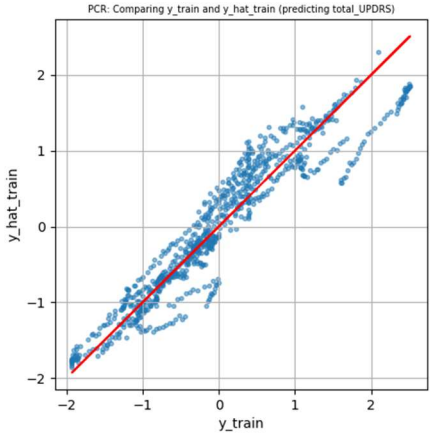


Figure 1- 72- PCR: scatter plot for train set predicting "total_ UPDRS"

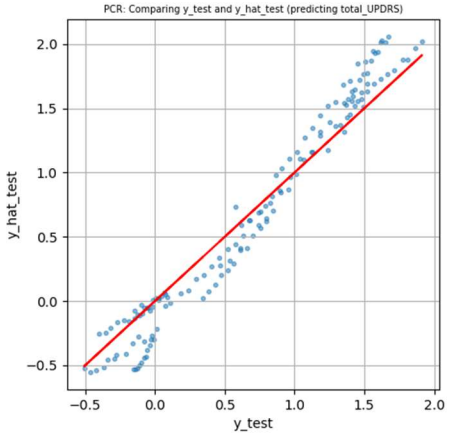


Figure 1- 73- PCR: scatter plot for test set predicting "total_ UPDRS"

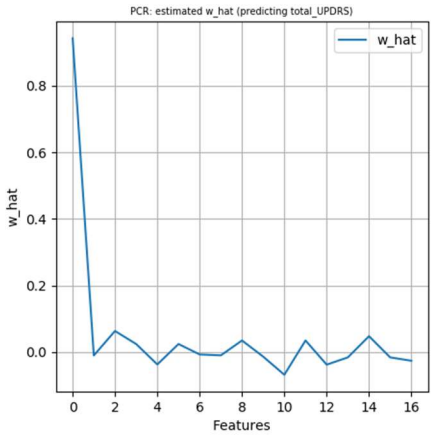


Figure 1- 74- PCR: estimated w for predicting "total_ UPDRS"

Figures 1-68 and 1-72 show that in train set the algorithm overestimated the "y" values in most of the samples; on the other hand, Figures 1-69 and 1-73 show that we have underestimation for higher "y" values in test set. From Figures 1-70 and 1-71 it is noticeable that the error range is higher in PCR and this is due to the selected features. In general, if we increase the percentage, we can have less errors in our prediction. But in case of having huge number of features the PCR can help us to reduce the

dimension of the problem. At the end from Figure 1-74 we can understand that the feature zero which is "motor_ UPDRS" has the highest weight among all the features.

1-4-6- K-Fold Cross Validation

The basic idea is to partition the dataset into K bins of equal subsets. In k-fold cross validation, the regression algorithms are executed for K times and in each time one of those k subsets is chosen as a test set and the rest K-1 bins are a train set. By comparing the squared error value for each K having similar performances will show us that the overfitting does not occur.

1-4-6-1- K-Fold Cross Validation Results

The k value has been set to be 5; therefore, the dataset is divided into 5 subsets by dividing the number of patients by 5. As there are 42 different case studies in the dataset, the subsets which are created are not equal in size. Table 1-14 shows the subsets in detail. There are three subsets with 8 patients and two with 9. Please note that in previous algorithms the test set was formed by having 6 patients and the train set included 36.

Table 1- 14- Size of Train set and Test set for each Fold

	K1	K2	K3	K4	K5
# Patients in Train set	34	34	33	34	33
# Records in Train Set	791	809	785	811	764
# Patients in Test set	8	8	9	8	9
# Records in Test set	199	181	205	179	226

As it is noticeable in Table 12, there are 5 different combinations of test set and train set with different sizes. For each dataset all the regression algorithm has been executed and the squared error value calculated. First, the results related to the prediction of "Jitter (%)" are shown.

Table 1- 15- Squared Errors for Train set and Test set when Predicting "Jitter (%)" using K-Fold approach

	K1	K2	K3	K4	K5
MSE Train Set Squared Error	3.68684983823	3.40584268994	3.49610973457	4.78321617797	3.80380787953
MSE Test Set Squared Error	0.48488897651	1.1161852239	0.786963691774	2.80493991166	3.61945812492
Iterative, Gradient Train Set Squared Error	3.69802814159	3.42123804861	3.50482615061	4.81291201642	3.81024129387
Iterative, Gradient Test Set Squared Error	0.48062082105	1.0974898396	0.787064676477	2.75657679291	3.49169892406
Steepest Descent Train Set Squared Error	3.69718217177	3.41960146431	3.50354528633	4.78862114968	3.80792524184
Steepest Descent Test Set Squared Error	0.483555941073	1.11471092355	0.789362407275	2.79637548022	3.65507746327
Ridge Regression Train Set Squared Error	3.69797475048	3.4208651814	3.50407772677	4.81268758779	3.80800303814
Ridge Regression Test Set Squared Error	0.48288605747	1.10213065968	0.792014200966	2.77418834382	3.65360697702
PCR Train Set Squared Error	3.87064513072	3.72043211863	3.69261786631	5.59643801301	4.16529296297
PCR Test Set Squared Error	0.542517162584	0.963176918729	0.771078761737	3.94542764389	3.02090522679

Table 1-15 shows the squared error for different regression algorithms in different combinations of dataset. Largest errors for both train set and test set are shown by red color. As it was predictable in most of the cases, the largest errors belong to the PCR algorithm. For better comparison between error results bar graphs have been provided which are shown from Figure 1-75 to 1-79. The squared errors are like each other in most of the cases. The largest error for train dataset belongs to k4 in all algorithms, and the largest error for test sets belongs to k5 except in PCR where k4 has the largest value for test dataset squared error. The smallest squared error between test dataset belongs to k1 in all runs. The interesting thing that can be concluded with these results is that having smallest squared error in train dataset does not always lead to a small squared error for test dataset. Also, it is possible to say that in all folds the value related to the squared error of the train dataset is larger than test cases' errors, which means that overfitting did not occur.

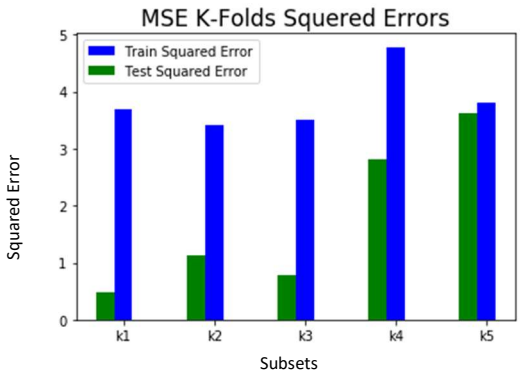


Figure 1- 75- MSE squared error predicting "Jitter (%)"

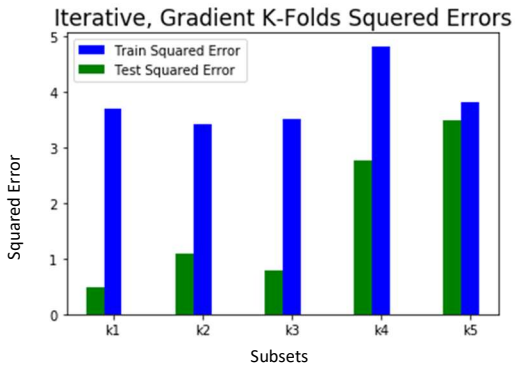


Figure 1- 76- Iterative gradient squared error predicting "Jitter (%)"

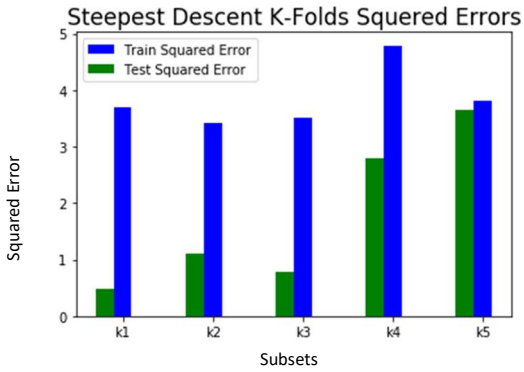


Figure 1- 77- Steepest descent squared error predicting "Jitter (%)"

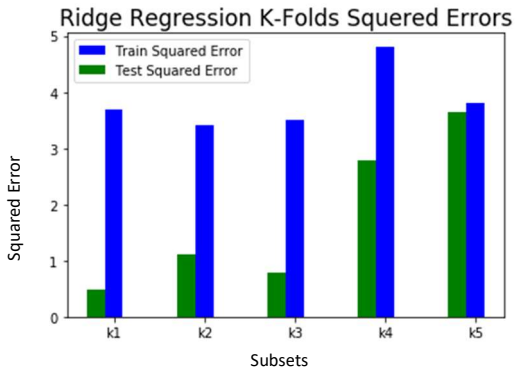


Figure 1- 78- Ridge regression squared error predicting "Jitter (%)"

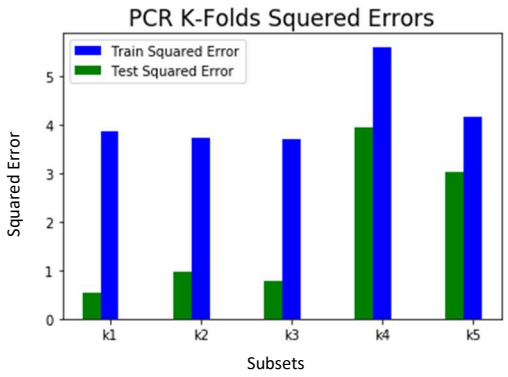


Figure 1- 79- PCR squared error predicting "Jitter (%)"

Like always the second step is to predict "total_ UPDRS" by using K-fold cross validation. Table 1-16 shows the squared error values for each algorithm by using K-fold cross validation method. This time in train set the largest squared error belongs to PCR. Surprisingly, in the test set most of the higher errors belong to MSE.

Table 1- 16- Squared Errors for Train set and Test set when Predicting "total_ UPDRS" using K-Fold approach

	K1	K2	K3	K4	K5
MSE Train Set Squared Error	78.9124518117	78.0298379857	68.249463253	61.4449623355	74.823539153
MSE Test Set Squared Error	12.0976375659	9.9340009023	46.6177770464	37.4115551559	42.9404096332
Iterative, Gradient Train Set Squared Error	78.9733074061	78.1110484803	68.3039629857	61.7041679131	75.1744152766
Iterative, Gradient Test Set Squared Error	12.0606929471	9.87624823257	46.3254776557	36.8124310591	42.102208416
Steepest Descent Train Set Squared Error	78.971689719	78.1088057162	68.3016609029	61.5566418203	74.8741779211
Steepest Descent Test Set Squared Error	12.0712743553	9.90220482706	46.3506184249	37.0898375706	42.7211247843
Ridge Regression Train Set Squared Error	78.9736551891	78.1115388262	68.305409774	61.7032053061	75.1759783674
Ridge Regression Test Set Squared Error	12.0590641909	9.86424490069	46.3255816043	36.8152927076	41.9498491286
PCR Train Set Squared Error	81.1096490834	80.4000660744	69.7878609672	64.1853279658	77.5387363772
PCR Test Set Squared Error	11.8838818543	9.55299749311	46.7077058224	36.0259519767	41.1403490403

By comparing Figures 1-80 to 1-84 it is easily understood that all the figures look like each other showing a common behavior. This time the smallest squared error value for train sets belongs to k4 and the minimum error value for test set belongs to k2. Furthermore, k1 has the maximum error value for train dataset and the largest squared error for test datasets belongs to k3.

At the end, it is necessary to mention that since there is no large error value for test sets and in all cases squared errors for test datasets are bigger than test datasets error and performance is similar in all the cases, then overfitting did not occur.

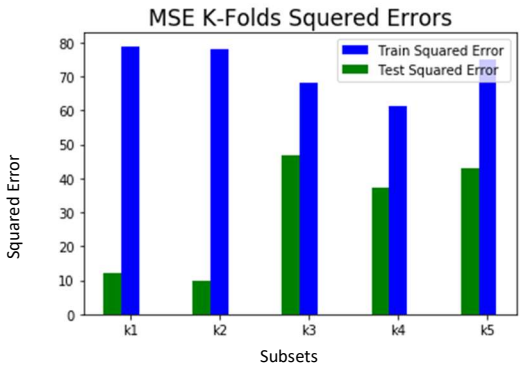


Figure 1- 80- MSE squared error predicting "total_ UPDRS"

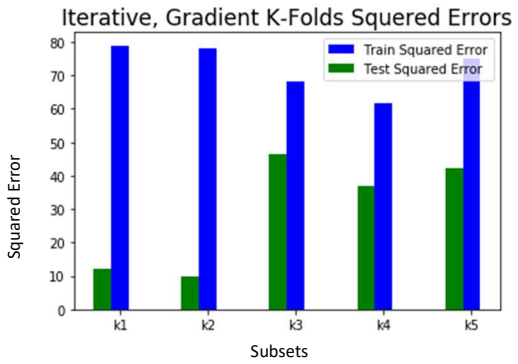


Figure 1- 81- Iterative gradient squared error predicting "total_ UPDRS"

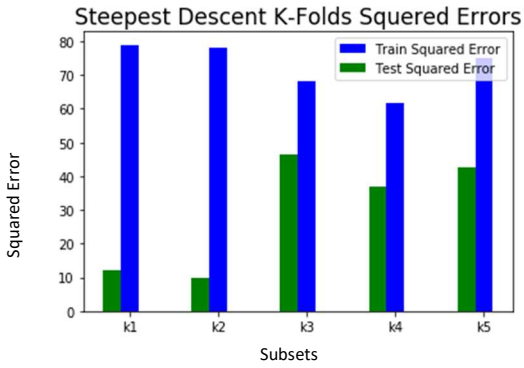


Figure 1- 82- Steepest descent squared error predicting "total_ UPDRS"

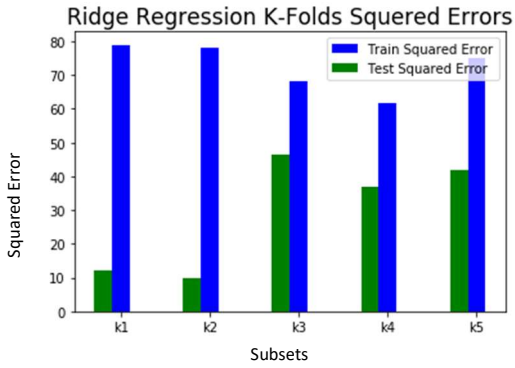


Figure 1- 83- Ridge regression squared error predicting "total_ UPDRS"

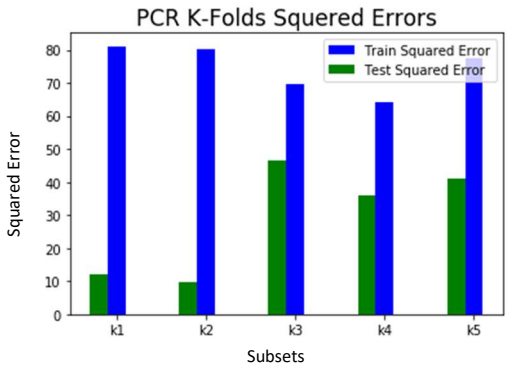


Figure 1- 84- PCR squared error predicting "total_ UPDRS"

1.5- Conclusions

By comparing the results and squared error values which are obtained for estimating "Jitter (%)" and "total_ UPDRS", it is possible to understand that the algorithms estimated "Jitter (%)" more accurately. Furthermore, according to the results that we obtained, we can say that the performance of the algorithms is similar to each other; however, each of them has its own advantages and disadvantages. Some of them are complex to implement and some of them have a time complexity and choosing which one to use depends on the dataset characteristic and the accuracy that we want to achieve.

2- Chronic Kidney Disease Dataset

2-1- Introduction

The kidneys job is to filter extra water and wastes from blood. Chronic kidney disease means that kidneys are damaged, and they cannot filter the blood in a way that they should. The kidneys have lost their functionality during a long period of time; therefore, the disease is called chronic. Early detection and therapy can avoid getting the situation worse. Many tests and exams are required to diagnose the disease. Sometimes it is difficult to diagnosis it in early phases; therefore, implementing machine learning approach could help doctors detect this kind of disease earlier.

2-2- Dataset

For the second experiments Chronic Kidney Disease dataset has been chosen as a case study. The objective of this lab is to work with decision trees. Using the dataset, we created and extracted hierarchical decision trees and used them to classify patients into two different groups, CKD[8] and not-CKD.

Table 2- 1- Chronic Kidney Disease Dataset

Index	Att	Description	Type	Metric
0	age	age	Numerical	age in years
1	bp	blood pressure	Numerical	bp in mm/ Hg
2	sg	specific gravity	Nominal	1.005, 1.010, 1.015, 1.020, 1.025
3	al	albumin	Nominal	0, 1, 2, 3, 4, 5
4	su	sugar	Nominal	0, 1, 2, 3, 4, 5
5	rbc	red blood cells	Nominal	normal, abnormal
6	pc	pus cell	Nominal	normal, abnormal
7	pcc	pus cell clumps	Nominal	present, not present
8	ba	bacteria	Nominal	present, not present
9	bgr	blood glucose random	Numerical	bgr in mgs/ dl
10	bu	blood urea	Numerical	bu in mgs/ dl
11	sc	serum creatinine	Numerical	sc in mgs/ dl
12	sod	sodium	Numerical	sod in mEq/ L
13	pot	potassium	Numerical	pot in mEq/ L
14	hemo	hemoglobin	Numerical	hemo in gms
15	pcv	packed cell volume	Numerical	
16	wc	white blood cell count	Numerical	wc in cells/ cumm
17	rc	red blood cell count	Numerical	rc in millions/ cmm
18	htn	hypertension	Nominal	yes, no
19	dm	diabetes mellitus	Nominal	yes, no
20	cad	coronary artery disease	Nominal	yes, no
21	appet	appetite	Nominal	good, poor
22	pe	pedal edema	Nominal	yes, no
23	ane	anemia	Nominal	yes, no
24	class	class	Nominal	ckd, notckd

[8] Chronic Kidney Disease

The Chronic Kidney Disease dataset is available on the University of California machine learning website[9]. It contains information related to 400 persons; 250 of them are classified as CKD (with the disease) and the rest classified as notCKD (without the disease). For each person 24 features are recorded. Table 2-1 shows the description of those attributes. Fourteen features are numerical with float type and 11 attributes are categorical with object type. Unfortunately, there are a lot of null values inside the dataset which are filled by "?".

2-3- Data Preparation

The first step is to transform categorical data to numerical one. One way is to replace them with 0 and 1 values. Table 2-2 shows the results of replacement for each categorical data.

Table 2- 2- replaced Values

Original Value	Replaced with
normal	0
abnormal	1
present	0
notpresent	1
yes	0
no	1
good	0
poor	1
ckd	1
notckd	0

In the next step empty space values like double space and tabs have been removed from the dataset. Furthermore, it is necessary to do something with the null values. For this step two solutions have been developed. First, it is possible to remove all the rows with the null values, and second, replace the null values with a specific number. For the second approach, since all the values in the dataset are positive, it is better to replace the null values with negative value like "-1" or "-0.5". At the end, features "class" is set as a target which means that the algorithm should predict this feature.

2-4- C4.5

C4.5 is one of the supervised approaches that can be used for classification. The algorithm uses decision trees as a classifier. C4.5 is based on the entropy's concept and information gain. It starts from all the data and splits it to subsets to create a hieratical tree. In machine learning, classification is used to predict if the new data belongs to one class or not. In this exercise the goal is to understand that the patient has the kidney disease or not.

In this lab, instead of implementing the C4.5, it is possible to use Scikit Learn[10] library which has a function that implements C4.5 for the hierarchical classification.

2-4-1- Classification Results

The first step is to remove all the data which include null values from the dataset and run the code. Figure 2-1 shows the result for this step.

[9] https://archive.ics.uci.edu/ml/datasets/chronic_kidney_disease
[10] http://scikit-learn.org/stable/index.html

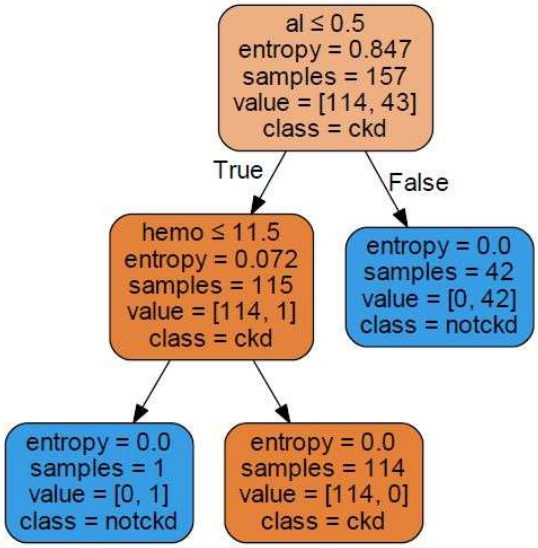


Figure 2- 1- Classification tree by removing null values

As it is noticeable, only 157 samples have remained among 400. Since 243 records included null values. As we can understand from Figure 2-1 and Figure 2-2 the classification tree includes only two features (please refer to Table 2-1 to see the features' indexes). It is noticeable that feature albumin (al) plays a big rule for classifying the data with entropy equal to 0.847. In 42 samples out of 157 the albumin value is greater than 0.5 and they are classified as notckd. Please not that albumin is a nominal data and it means 42 patient had albumin value not equal to zero. Hemoglobin (hemo) is the second feature that has an impact on the classification tree. This attribute has numerical values. This time one sample out of 115 is classified as a healthy person which has the hemoglobin level less than 12. In general, 43 persons have been classified as healthy and the rest have the kidney disease.

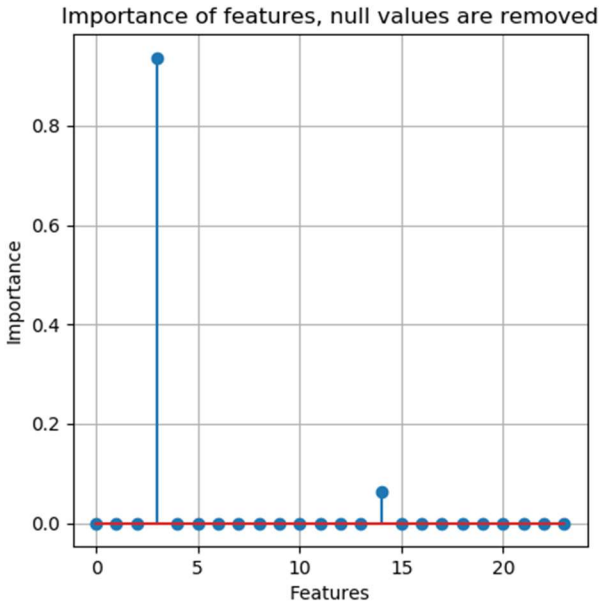


Figure 2- 2- Features' Importance, removing null values

In the second step the null values have been replaced by "-1". Figure 2-3 shows the classification tree. Now the result is completely different, and the classification tree has more branches. In this step, more features are used to create the tree.

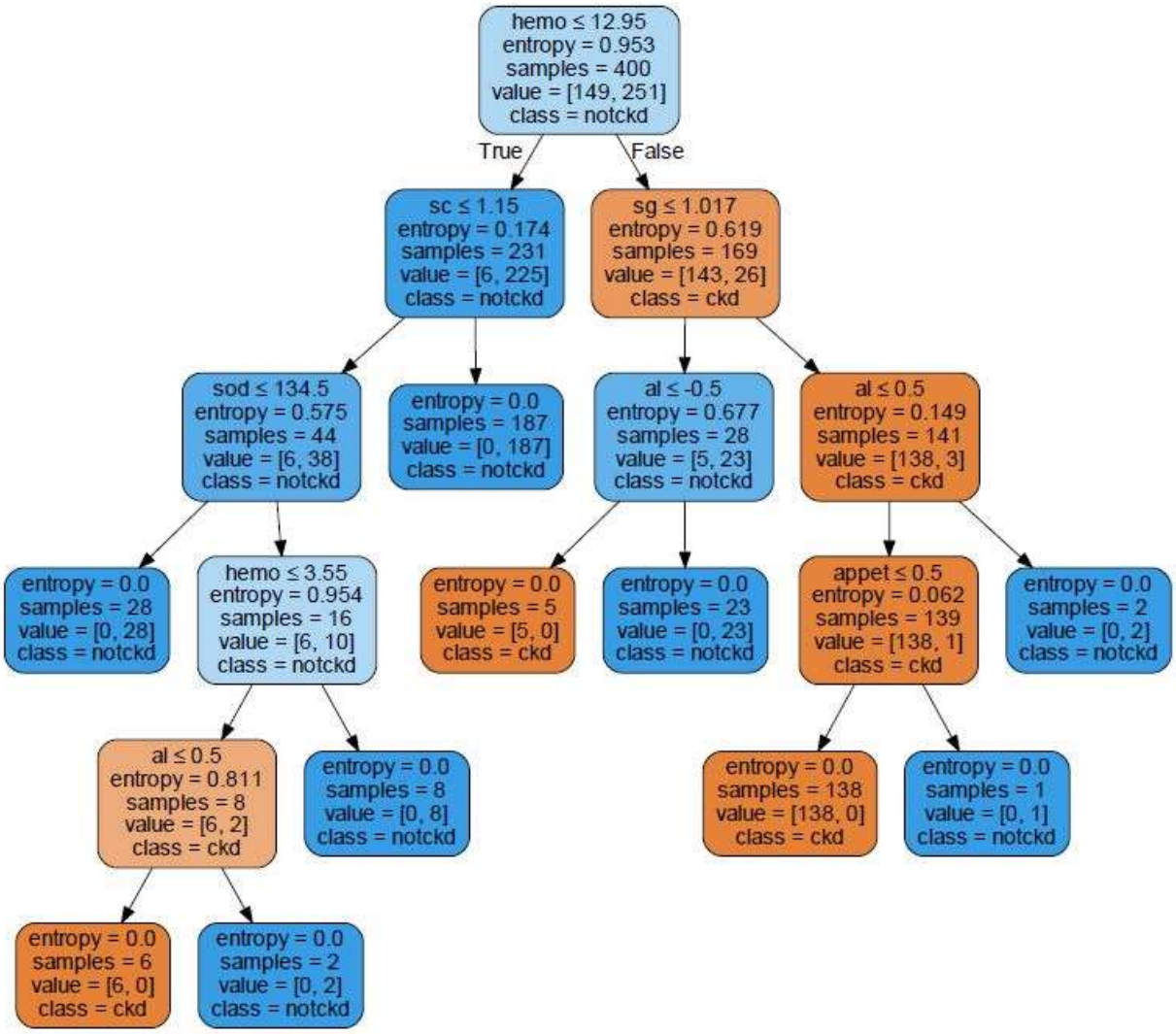


Figure 2- 3- Classification tree by replacing null values with -1

Figure 2-4 shows the futures' importance (please refer to Table 2-1 to see the features' indexes). This time feature Hemoglobin is the most important feature. Furthermore, specific gravity, albumin, serum creatinine, hypertension and sodium have the highest entropy values.

In conclusion, it is possible to say that choosing the way to deal with the null values has a huge impact on creating classification tree. Removing the null values will simplify the problem that could lead to having error using this tree to classify the other data. Also replacing them with other values may affect the rules in creating dissection tree for example in Figure 2-3 in the second level the algorithm is developing the branches by considering the albumin less than or equal to (-0.5) which means that the null value is involved in the decision.

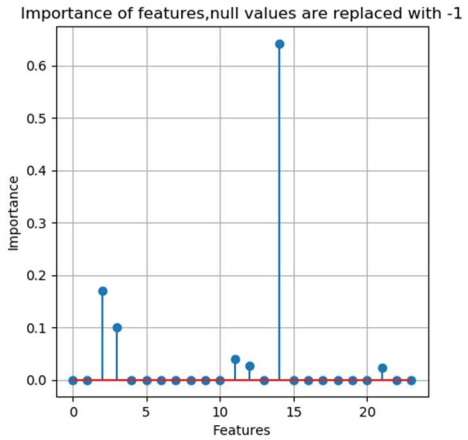


Figure 2- 4- Features' Importance, Replacing null values with "-1"

This result shows the importance of having data with a good quality in using the algorithm to create a better decision tree.

2-5- Agglomerative Clustering

Clustering is unsupervised method. This method clusters the data in "K" subsets base on their similarities therefore, the basic idea is to make sure that if you have two nearby points they end up in the same cluster.

Agglomerative clustering is typically bottom-up method. The way that it works is that it starts by having a collection of singleton clusters so for each instance of the data it creates a cluster that contains just that instance line so there is c_i that contains $\{x_i\}$. Then it iteratively does the following procedure. It will find the pair that is closest to each other by calculating the minimum distance between to clusters, merge them and create a new cluster, delete the old two clusters and repeat this until there is only one cluster left.

2-5-1-Clustering Results

For implementing agglomerative clustering, it is possible to use SciPy[11] library in python. This library by default uses the single methods also known as the nearest point algorithm for calculating the distances between clusters (Equation 2- 1).

$$d(u, v) = min(dist(u[i], v[j]))$$

Equation 2- 1- Nearest Point

Figure 2-5 shows the result of agglomerative clustering. Please note that for creating this plot, single method is used to calculate the distances.

[11] https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html

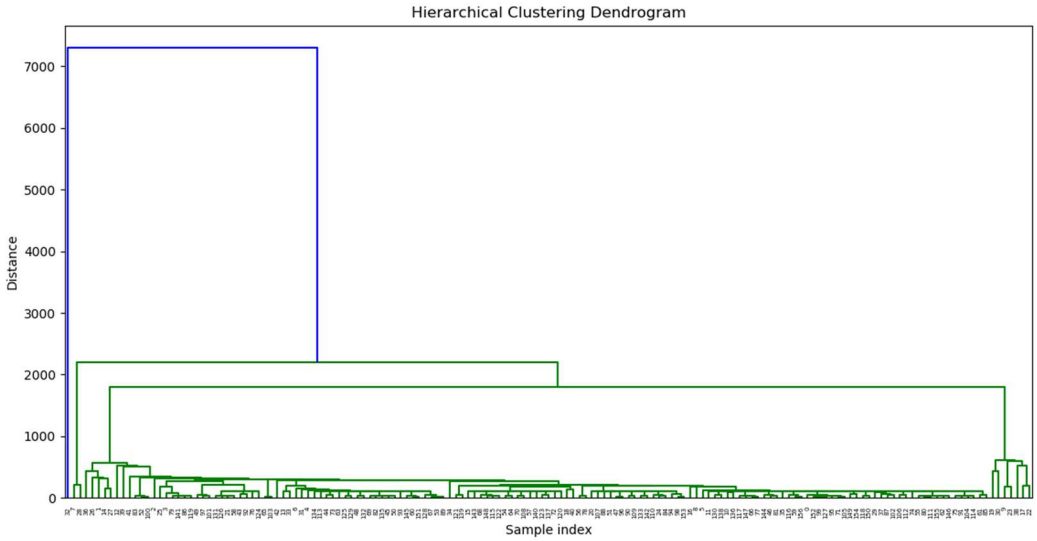


Figure 2- 5- Agglomerative clustering- Dendrogram.

3- Moles Dataset

3-1- Introduction

Moles are very common, and many people have one or more on their skins. There are many types of moles with different sizes and colors. Moles usually emerge in childhood and adolescence, nevertheless if a new mole appears in older ages or it changes the size and color that could be a sign of melanoma cancer. In this case, the clinical examination is necessary since the early detection of skin cancer helps to control the situation better.

To diagnose melanoma, doctors take into account five different features: Asymmetry, Borders, Color, Diameter and Evolution (ABCDE). The goal of this lab is to help doctors analyze the borders using K-means algorithm. Also, the aim is to calculate the ratio between perimeter of the moles border with the perimeter of the circle having the same area of the moles. The moles with the higher ratio have a high probability of being cancerous.

3-2- Dataset

The dataset that we used for this laboratory has 55 different moles' images with size of 584x583 pixel. This set includes 11 pictures of moles labeled as low risk, 16 images considered as medium risk and finally 27 pictures as belonging to different melanomas.

3-3- K-means Algorithm

K-means is one of the classical clustering algorithms. It is one of the unsupervised methods which means that the data has not been labeled. This method clusters the dataset into "K" different clusters. It chooses the center of clusters randomly. Furthermore, it assigns all the instances to the nearest cluster center. In the next step, the algorithm calculates the mean value for each cluster. The previous clusters' centers will be replaced by these new values. At the end, the whole method will be repeated by choosing new random values as clusters' centers. The algorithm continues until the same points are calculated for each cluster in two back to back rounds. In this case, the center for each cluster is confirmed and it is not going to change anymore.

In this laboratory we used K-means available in Scikit-learn library[12].

3-4- Algorithm

The steps that we need to take are as follows:

First, we need to read the image and apply the K-means algorithm on it with the number of clusters set to 3. K-means will cluster the images into three different colors. In this way each cluster's center will have one specific color.

Among the centroids the darkest color corresponds to the moles. Therefore, in the second step it is necessary to find the darkest color among the three different clusters.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Furthermore, the median of the region with the darkest color should be calculated. Probably this median is in the center of the mole or it is close to it.

The following step is to start from the center of the mole and find the square that includes all the mole and crop the image to generate a sub- image having the mole only, not the other parts of the original image. The algorithm starts from the median, goes to the top, down, left and right of the center step by step until the color of the pixel changes from the darkest to a different color for 10 consecutive pixels, Then it calculates the distance from that position with respect to the median and chooses the largest one among up, down, right and left pixels. In this way, we are able to find the edges of the square that we need.

Finally, the columns and rows are considered one by one to find the indexes of the first pixel and last pixel of each column and row. In this way, we are able to find the contour including the moles border, and at the end we need to plot the contour.

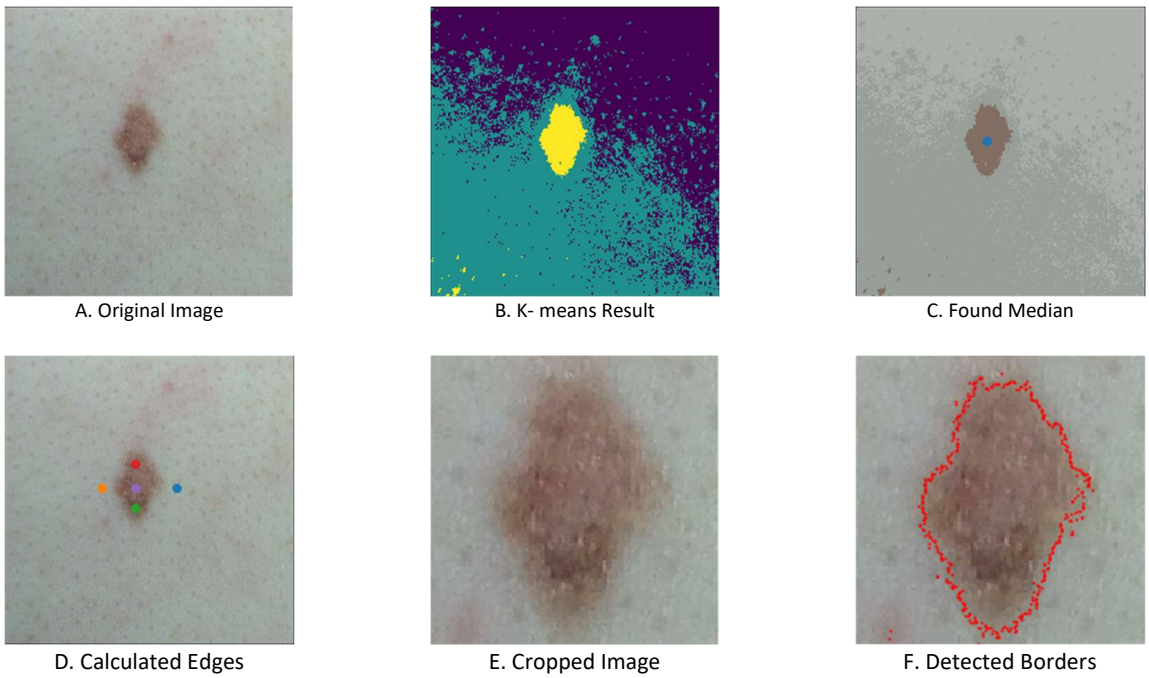
3-4-1- Algorithm Results

Table 3-1 shows the mean values for the calculated ratios. As it is noticeable, the mean value for melanoma is higher than the other ones which is what we expected. However, the values are close to each other.

Table 3- 1- First approach's mean values

Moles type	Mean Ratio
Low Risk	1.0667574360618055
Medium Risk	1.106592565198795
Melanoma	1.1449512752765667

Let us check the results in more details. Figure 3-1 shows the first approach's steps applied on image.



A. Original Image | B. K- means Result | C. Found Median

D. Calculated Edges | E. Cropped Image | F. Detected Borders

Figure 3- 1- First Suggested Algorithm Steps

In “low_risk_1”, the final result is good but not good enough; the red border line is not continuous. This is biased and some improvement on the algorithm is needed. Some other problems have also been detected. For example, Figure 3-2 shows that in some images the calculated median is not in the center of the mole due to some pixels clustered with the same dark color on the other part of the image.

Figure 3- 2- Calculated Median for Image “low_risk_4”

The second problem is that in some moles the algorithm was not able to find the edges correctly to crop the image since some moles had unusual shapes. These mostly belonged to the cases that were labeled as melanoma. This problem is shown in Figure 3-3.

A. Calculated Cutting Points in “melanoma_24”

b. Final Result for “melanoma_24”

Figure 3- 3- Problem in Calculating the Edges in melanoma_24

Figure 3-4 shows another problem detected in “melanoma_23”. The shape of the mole is irregular. Therefore, the algorithm was not able to find the borders correctly. Moreover, in this image some hair is detected too.

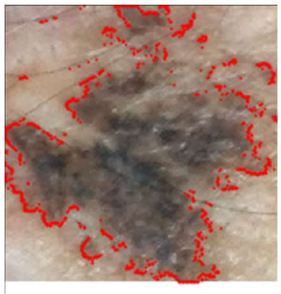


Figure 3- 4- Borders founded for “melanoma_23”

3-5- Improving the Algorithm

To improve the algorithm, the first thing that we tried to change was to improve the approach that we used to find the contour. This time instead of considering first and last dark pixels in columns and rows

the algorithm goes one by one and finds the neighbors of each pixel and according to the color of the neighbors it will decide to consider a given pixel as the border of the mole or not.

The new algorithm works in this way:

First it will create a matrix with the same size as the cropping image and set all values to zero. After this, it will calculate the indexes of the pixels with dark colors and then set the corresponding elements in the matrix to one.

In the next step the algorithm goes through cells one by one and calculates the summation of the values of neighboring cells. If the results become zero, it means that the matrix cell is located outside the mole (Figure 3-5) and if the value of the cell itself is one the algorithm will change it to zero (Figure 3-6).

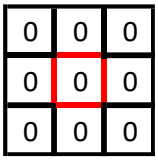


Figure 3- 5- The Cell is located outside the Mole

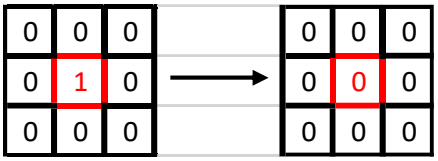


Figure 3- 6- Change the value from one to zero if all the neighbors are zero

If the summation of the neighbors becomes equal to eight, it means that the pixel is located inside the mole (Figure 3-7). Therefore, if the value of the pixel is zero, the algorithm will change it to 1 (Figure 3-8).

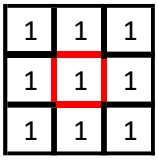


Figure 3- 7- Th cell is inside the Mole

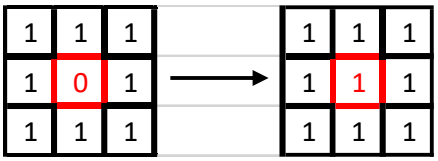


Figure 3- 8- Change the value from zero to one if all the neighbors are one

Finally, if the summation of the neighbors becomes more than zero, the cell will be considered as border of mole (figure 3-9).

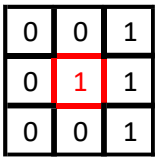


Figure 3- 9- The cell will be considered as border

In this way, some small noises were removed from the matrix. Table 3-2 shows the results after applying this new method on the dataset. Comparing this table with Table 3-1 it is possible to understand that the results are higher and also the difference between the different types of moles were high.

Table 3- 2- Mean values after first improvements

Moles type	Mean Ratio
Low Risk	1.9949654431870352
Medium Risk	2.421365043536181
Melanoma	3.6183518078848995

Figure 3-10 shows the new generated borders for "melanoma_23" which shows a higher accuracy in detecting the borders. Nevertheless, some areas are detected as border outside the mole and inside the mole.

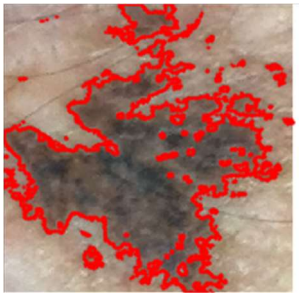


Figure 3- 10- Borders founded for "melanoma_23" after first improvement

The next step to improve the algorithm is to calculate the bigger square for cropping the image. Therefore, a constant value was added to the maximum distance calculated from the median. In this way, we could make sure that all the mole will be included inside the cropped image. However, some dark pixels that are not part of the mole will also be added to the final cropped image. Table 3-3 shows the calculated mean value after adding this step to the previous algorithm. From this result we can notice that the mean values are increased a little bit.

Table 3- 3- Mean values after second improvements

Moles type	Mean Ratio
Low Risk	2.297399186671033
Medium Risk	2.577279661760072
Melanoma	3.908491347306002

Figure 3-11 shows the new detected border for "melanoma_24". This time all the mole was in included inside the cropped image.

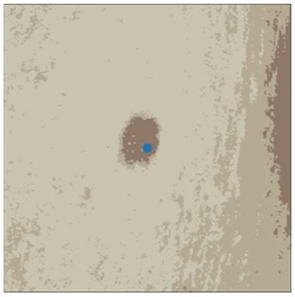
Figure 3- 11-Borders of "melanoma_24"

The next possible step to improve the algorithm is to consider the center of the image as the center of mole instead of calculating the median. This approach works well for most of the cases but not for all the images. Now we should make sure that the mole is located in the center of the image; otherwise, this approach is useless. Table 3-4 shows the result for this method with a huge impact on the calculated mean values.

Table 3- 4- Mean values after third improvements

Moles type	Mean Ratio
Low Risk	2.8121796924483955
Medium Risk	3.3754129172938985
Melanoma	4.283051464660765

Figure 3-12 shows the mole "low_risk_4" this time the algorithm was able to find the borders of the mole.



A. Center of "low_risk_4"

b. Final Result for "low_risk_4"

Figure 3- 12- "low_risk_4" detected borders

The final step for improving the algorithm is to filter the images to remove the noise from it. Accordingly, OpenCv[13] library (Open Source Computer Vision Library) was imported into the project. Three different filters were used in this step, blur[14], median blur[15]and gaussian blur[16]. Using these filters led to smooth images.

Table 3- 5-Final Mean values after all improvements

Moles type	Mean Ratio
Low Risk	1.087290825733587
Medium Risk	1.793617141016362
Melanoma	1.956214583935656

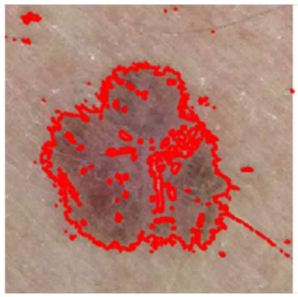
Table 3-5 shows the final results. This time the mean values are less than the previous time since some noises were removed from the image and the number of pixels were detected inside the mole and outside of it as borders were decreased. However, as we expected the ratio for melanomas images was higher than the other ones. Figure 3-13 shows the results for some moles before and after applying filters on them.

[13] https://opencv.org/

[14] https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=blur#blur

[15] https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=medianblur#medianblur

[16] https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#gaussianblur



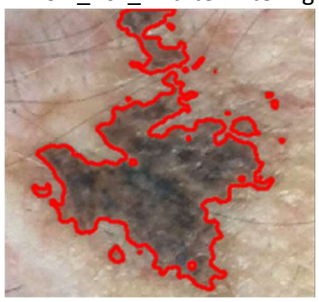
A. “Melanoma_10” before filtering

B. “Melanoma_10” after filtering

C. “Low_risk_1” before filtering

D. Low_risk_1” after filtering

E. “melanoma_23” before filtering



F. “melanoma_23” after filtering

G. “low_risk_4” before filtering

H. “low_risk_4” after filtering

Figure 3- 13-Final Results of before and after filtering the images

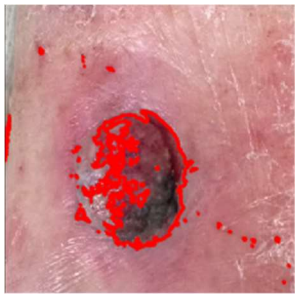
As it is noticeable in Figure 3-13, the algorithm is able to detect the borders more accurately with less noise.

From the above pictures, it is obvious that the final algorithm is working much better than the first one. Nonetheless, the final method is not working well on all the images, especially “melanoma_27” shown in

Figure 3-14, since it seems that there is a big shadow on most of the image, or in the image related to

Figure 3- 14-Final result for "melanoma_27"

"melanoma_4" shown in Figure 3-15 some part of the melanoma is brighter than the other parts. As a result, the algorithm is not able to find the exact borders for it. For these reasons more improvement is needed.



A. "melanoma_4" before filtering

B. "melanoma_4" final result

Figure 3- 15- "melanoma_4" founded borders before and after filtering

Some suggestion is two provide images with better quality, with good lights, locate the moles in the center and avoid having shadows in the picture. Furthermore, it is desired to use better filters to process the image in a better way.

4- Arrhythmia Dataset

4-1- Introduction

When the electrical signals to the heart that harmonize heartbeats are not working well, an arrythmia happens. In general, arrythmia describes an unusual heartbeat, the heart may beat too fast, too slowly, too early or irregularly.

In this laboratory we were asked to work an Arrhythmia dataset to classify the patients in different groups. The classification can be binary or multi-class. In the binary classification the aim is to classify the patients in two different groups, having Arrythmia or not. Furthermore, in multi-class approach the goal is to divide the patients in different groups according to the Arrythmia level that they have.

Different approaches and methods have been implemented in this laboratory and the performance of them have been analyzed and the sensitivity and specificity have been calculated.

4-2- Dataset

The arrythmia dataset which is chosen for this lab is available on University of California machine learning website[17]. This dataset contains 452 individual patients and for each person 279 attributes have been recorded. Furthermore, the dataset is divided into 16 different classes and each class belongs to a type of Arrythmia. Table 4-1 shows these categories and the number of records for each class. Nothing is recorded for classes 11,12 and 13.

Table 4- 1- Class distribution of Arrythmia's dataset

Class Code	Class	Number of instances
1	Normal	245
2	Ischemic changes (Coronary Artery Disease)	44
3	Old Anterior Myocardial Infarction	15
4	Old Inferior Myocardial Infarction	15
5	Sinus tachycardy	13
6	Sinus bradycardy	25
7	Ventricular Premature Contraction (PVC)	3
8	Supraventricular Premature Contraction	2
9	Left bundle branch block	9
10	Right bundle branch block	50
11	1. degree AtrioVentricular block	0
12	2. degree AV block	0
13	3. degree AV block	0
14	Left ventricule hypertrophy	4
15	Atrial Fibrillation or Flutter	5
16	Other	22

In this lab we are going to implement binary classification and multiclass classifications. In the first case, the dataset will be divided into two groups, one for the patients with the disease and the other for those without the disease. Therefore, the target is to just predict that the record belongs to which group. In

[17] http://archive.ics.uci.edu/ml/datasets/Arrhythmia

multiclass classification, the aim is to divide the dataset into 16 classes which refer to different kinds of Arrythmia and the goal is predict which category each patient belongs to.

4-3- Data Preparation

Unfortunately, the dataset includes some missing values shown as "?". Moreover, some columns only store zeros which means that they do not have any information. All these columns had to be excluded from the dataset. At the end, 452 patients with 258 attributes remained.

4-4- Results Validation

In this lab we needed to check the accuracy of the results and how well they could classify the data. To this end, we calculated sensitivity and specificity in binary classification and created a confusion matrix in multiclass classification.

4-4-1- Sensitivity and Specificity

To validate our results in binary classification we needed to do some calculations. Table 4-2 shows the metrics that we needed to measure.

Table 4- 2- Validation Metrics

Metric	Description
True positive (TP)	Sick people correctly identified as sick, correctly identified
True negative (TN)	Healthy people correctly identified as healthy, correctly rejected
False positive (FP)	Sick people incorrectly identified as healthy, incorrectly rejected
False negative (FN)	Healthy people incorrectly identified as sick, incorrectly identified
Sensitivity	True positive rate (TPR), True positive/ Total sick patient (P)
Specificity	True negative rate (TNR), True negative/ Total healthy patient (N)
Accuracy (ACC)	(TP + TN) / (P + N)

The values of sensitivity and specificity are between zero and one. For a good classification higher values of sensitivity and specificity are needed, and we targeted at one.

4-4-2- Confusion Matrix

In multiclass classification, to describe the performance of the algorithm we need to create a confusion matrix. In this matrix, each element of "c_{ij}" is the probability that a patient is classified in a class "j" while the true class is "i". Since we are talking about probabilities, the summation of elements in each row should be equal to one. The confusion matrix of the algorithm with 100% accuracy will be one on the diagonal and zero for the rest of the elements.

4-5- Minimum Distance Criterion

The idea behind the Minimum Distance Criterion (MDC) is to compare the distance between the features of new record and the average of features of each class. Therefore, the new record will be assigned to the class which has the smallest distance with respect to the features of the new instance. The distance will be calculated by using Equation 4-1.

$$R_j = \{u : ||u - x_j||^2 \leq ||u - x_k||^2 , \forall k \neq j\}$$

Equation 4- 1-: Minimum Distance

In this formula "u" is the vector containing the features values of new record, and "x_j" and "x_k" are the vectors with the mean of the features of class "j" and "k" which correspond to the two classes: with the disease or healthy. Accordingly, in the binary classification the distances should be calculated with respect to the two classes and in the multi classification the distance between new patient record and all the classes should be calculated one by one.

4-5-1- MDC Binary Classification Results

Table 4-3 shows the results for MDC binary classification.

Table 4- 3- MDC Binary classification results

Metric	Values
True positive (TP)	131
True negative (TN)	178
False positive (FP)	67
False negative (FN)	76
Sensitivity	0.632850241545
Specificity	0.726530612244
Accuracy (ACC)	0.683628318584

According to Table 4-3 the sensitivity and specificity values are lower than one and the accuracy of the algorithm is 68%. Therefore, the performance of the classifier is not good enough.

4-5-2- MDC Multiclass Classification Results

Figure 4-1 shows the confusion matrix for MDC. As it is noticeable, the higher values are on the diagonal and only for classes 8, 9 and 15 the values are equal to one.

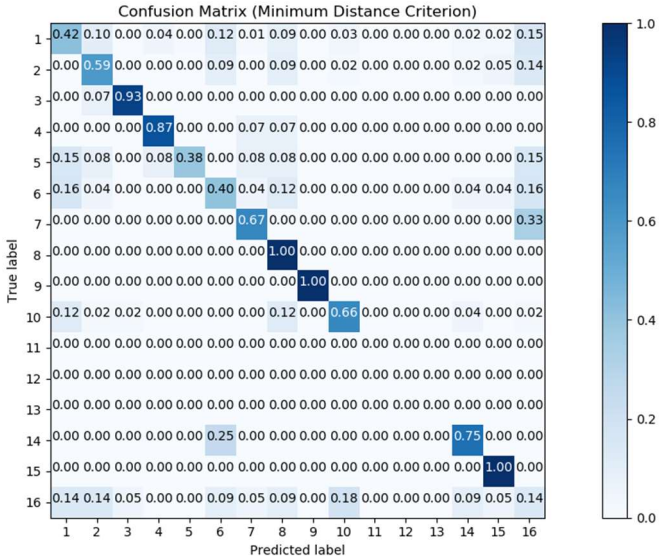


Figure 4- 1- MDC Confusion Matrix

4-6- Bayes Criterion

Here, again we are going to classify the dataset in two different groups; therefore, we have two hypotheses, "H_1": being healthy and "H_2": being sick (Equation 4-2).

$$H_1: y = x_1 + v_1$$

$$H_2: y = x_2 + v_2$$

$$P(H_1) \neq P(H_2)$$

Equation 4- 2- Hypotheses

In this formula, "y" is the set of measured attributes for the patient that we want to classify, "x_1" is the mean of "y" for the patients that have been classified as healthy by doctors which means "H_1" is true. On the other hand, "x_2" is the mean of "y" for the persons that have been classified as sick. Therefore, this time "H_2" is true and we assume that "v" is the vector of statistically independent Gaussian random variable. Now by using Bayes rule (Equation 4-3) we can evaluate Equation 4-4 and the person will be assigned to the class that has the highest probability.

$$P(A|C) = \frac{P(C|A)P(A)}{P(C)}$$

Equation 4- 3- Bayes Rule

$$P(H = H_k | y = y(n)) \quad k = 1, 2$$

Equation 4- 4

In other words, it is possible to say that, if the probability of being healthy given "y = y(n)" becomes bigger or equal to the probability of being sick given "y = y(n)", then the person will be classified as healthy. Otherwise, he or she will be assigned to sick class.

At the end it is necessary to mention that in this case as the covariance matrix is not convertible, Principal Component Analysis was performed in order to reduce the number of features and only keep the most important ones with higher eigenvalues.

4-6-1- Bayes Criterion Results

Since we are using PCR to select features with higher eigenvalues, different percentages were used to estimate the results.

Table 4- 4- Bayes criterion results

PCR (0.90)		PCR (0.95)		PCR (0.99995)	
Metric	**Values**	**Metric**	**Values**	**Metric**	**Values**
True positive (TP)	56	**True positive (TP)**	68	**True positive (TP)**	176
True negative (TN)	235	**True negative (TN)**	239	**True negative (TN)**	244
False positive (FP)	10	**False positive (FP)**	6	**False positive (FP)**	1
False negative (FN)	151	**False negative (FN)**	139	**False negative (FN)**	31
Sensitivity	0.27053140096618356	**Sensitivity**	0.3285024154589372	**Sensitivity**	0.8502415458937198
Specificity	0.9591836734693877	**Specificity**	0.9755102040816327	**Specificity**	0.9959183673469387
Accuracy (ACC)	0.6438053097345132	**Accuracy (ACC)**	0.6792035398230089	**Accuracy (ACC)**	0.9292035398230089

Tables 4-4 shows the results. As it is noticeable, by increasing the PCR percentage the results become better and better and by comparing the results with MDC, we can conclude that Bayes criterion works better than MDC. Please note that here we did not divide the data set into train set and test set and the results belong to the whole dataset. To have a better observation, it is good to run these algorithms with new samples to see how they behave with new patients and evaluate them in a better way.

4-7- Neural Networks

An Artificial Neural Network (ANN) is an information processing model that is trying to copy the performance of animal nervous system including the functions of the brain and nerves and information processing capability. The main section of this model is the modern structure of the data processing system. It is a collection of a huge number of highly inter-connected processing elements (neurons) working together to solve a specific problem. Imagine the ANN as a kind of a black box. The input layer which takes one or multiple inputs, processes them by using hidden layers and sends them to the output layer which can be one or multiple outputs. The ANN itself consists of many small units called neurons. These neurons are grouped into several layers. Neurons of one layer interact with the neurons of the next layer through weighted connections which really adjust connections with a real valued number attached to them. A neuron takes the value of a connected neuron and multiplies it with its connections weight. The sum of all connected neurons and the neurons bias value is then put into so called activation function which simply mathematically transforms the value before finally it can be passed onto the next neuron. This way the inputs are propagated through the whole network that is pretty like all the network does, but the real deal behind the neural network is to find the right weights in order to get the right results.

For this laboratory we were asked to use TensorFlow[18] library which is one of the powerful open source machine learning frameworks designed by Google and developer job is to design the structure of ANN which is explained in lab description by the professor.

We used ANN for two different classifiers, first we used it for binary classification and then for multiclass classification.

4-7-1- ANN Binary Classification Results

For this laboratory, the database has been divided into train set and test set.

Table 4- 5- ANN Binary classification train set results

Metric	Values
True positive (TP)	83
True negative (TN)	126
False positive (FP)	1
False negative (FN)	16
Sensitivity	0.8383838383838383
Specificity	0.9921259842519685
Accuracy (ACC)	0.9247787610619469

Table 4-5 shows the results for train set and Table 4-6 shows the results for test set. By comparing these two tables, we can notice that the ANN worked very well in the training phase with accuracy close to 92%. On the other hand, in the test phase the accuracy dropped to near 61% which is a little bit low. We can conclude that the algorithm adopts quite well on the train set which performed overfitting. Therefore, the performance over the test set was not good enough. Please note that the gradient descent optimizer was set to 0.0005 which was found by trial and error and the number of iterations was set high at 200000, and the algorithm spent a lot of time to estimate the final results.

[18] https://www.tensorflow.org/

Table 4- 6- ANN Binary classification test set results

Metric	Values
True positive (TP)	57
True negative (TN)	83
False positive (FP)	35
False negative (FN)	51
Sensitivity	0.5277777777777778
Specificity	0.7033898305084746
Accuracy (ACC)	0.6194690265486725

4-7-2- ANN Multiclass Classification Results

For this part, the gradient descent optimizer was set to 0.005 which was found by trial and error and the number of iterations was set very high at 700000 and this made the algorithm to spend a lot of time to calculate the results.

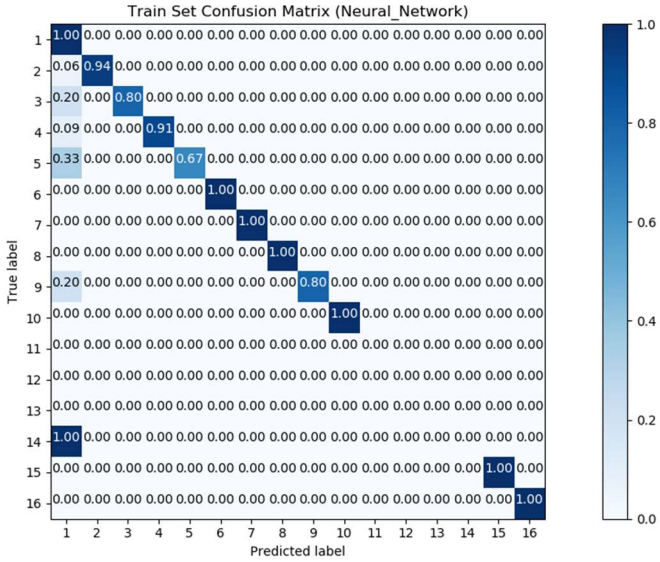


Figure 4- 2-Confusion Matrix for ANN train set

Figure 4-2 shows the result for the train set. The ANN was able to classify the data in a good way for most of the classes. It predicted the label for classes 1, 6, 7, 8, 10 15 and 16 with complete accuracy but it did not estimate the true label for class 14.

The results for the test set are shown in Figure 4-3. Like the binary classification again the ANN method did not work well on the test set and it is noticeable that instead of having the diagonal confusion matrix like the train set, most of the data are misclassified as being class 1 which is not right. This may be because of the way the ANN structure had been designed. because of the parameters that we selected for the number of iterations and the gradient descent optimizer or the dataset characteristics.

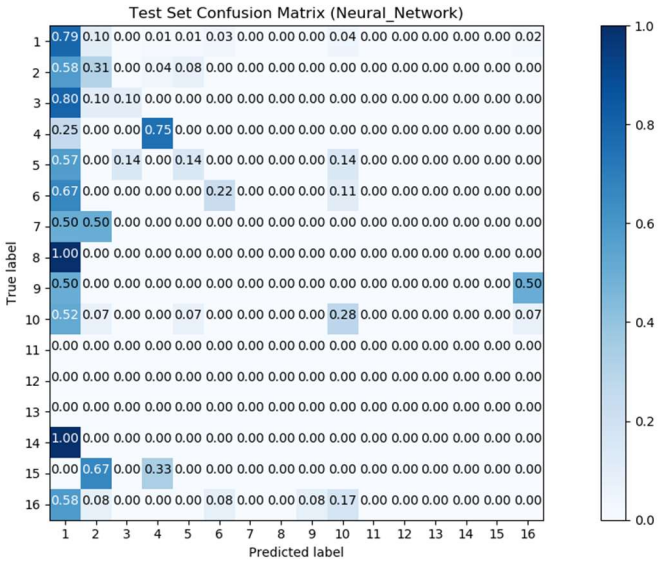


Figure 4- 3- Confusion Matrix for ANN test set

4-8- Support Vector Machine

One of the supervised machine learning algorithms is Support Vector Machine (SVM). The idea behind the SVM is to find a hyperplane in the F-dimensional space (F is the number of features) that distinctly classifies the data points. Data points can belong to two different classes: class "-1" or class "1" To separate the two classes of the data points, there are many available hyperplanes that can be selected. The objective is to choose the one that can maximize the margin, that is, the distance between data points of both classes.

Equation 4-5 is the one that can be used to separate the two classes.

$$x^T w - b = 0$$

Equation 4- 5

The hyperplane is identified by vector "w" and the "b" is the offset. By using Equation 4-5 we will have two hyperspaces, bigger than zero for class"1" and smaller than zero for class"-1" and the new data points will fall in one of these hyperspaces.

One thing that should be taken into consideration in SVM is the value of the relaxation parameter "C", also called box constrain. If the relaxation parameter is set to a large value, the relaxation will decrease, and the solution will tend to generate overfitting.

In this lab instead of implementing the SVM by ourselves, we used the Scikit Learn version.

4-8-1- SVM Results

This time again the dataset is divided in two equal subsets to be used as train set and test set. Furthermore, like section 4-5, PCA is used to select features with higher eigenvalues; also the relaxation parameter has been set to .0005. This value was found by trial and error.

Table 4-7 shows the results for training set and Table 4-8 shows the results for test set.

Table 4- 7- SVM Binary classification train set results

Metric	Values
True positive (TP)	81
True negative (TN)	117
False positive (FP)	10
False negative (FN)	18
Sensitivity	0.8181818181818182
Specificity	0.9212598425196851
Accuracy (ACC)	0.8761061946902655

If we compare the two tables, we will realize that the SVM works well in train set and not well enough in test set. The False negative rate is high in test set. Since we are working with patients' data classifying a sick person as a healthy one has a high risk and can put the person's life in danger.

Table 4- 8- SVM Binary classification test set results

Metric	Values
True positive (TP)	68
True negative (TN)	84
False positive (FP)	34
False negative (FN)	40
Sensitivity	0.6296296296296297
Specificity	0.711864406779661
Accuracy (ACC)	0.672566371681416

5- Voice Dataset of People with Parkinson

5-1- Introduction

Parkinson's disease (PD) is a nervous system disorder that grows slowly step by step in a long term and affects movements. The cause of the PD is unknown, but it makes certain nerve cells (neurons) in the brain break down or die little by little.

In this laboratory we were asked to work with voice record of people with Parkinson and voice of healthy people to analyze the voices with Hidden Markov Models (HMM) in order to classify them as healthy or sick.

5-2- Dataset

The data set that has been chosen for this laboratory has two parts; the first one is 10 audio records of voices belonging to healthy people. On the other hand, the second part contains voice records of 10 patients affected by PD.

5-3- Preprocessing

First, we need to convert the voices into vectors. Each record has a sampling frequency equal to 44.1 kHz which is possible to be reduced by 5 times. Furthermore, it is necessary to normalize the data to have a mean value equal to one. After this, we need to find the peaks in the voice signal and divide the space between each two peaks uniformly into seven subspaces. For this laboratory we need to use only the first 2000 samples of each record. At the end, since the HHM that we are going to use is only able to manage finite numbers it is necessary to quantify the voice samples.

After preprocessing, it is possible to create train and test sets. In each set 5 records belong to healthy people and 5 records belong to sick people having the PD.

5-4- Hidden Markov Models

The Markov model is discrete time random processing model used to represent the system that changes randomly. This model has a special characteristic shown in Equation 5-1 which is known as a Markov property, which means that the state "x_n" depends only on the state "x_{n-1}" not on all the events that occurred before that.

$$P(x_n|x_{n-1}, x_{n-2}, \dots) = P(x_n|x_{n-1})$$

Equation 5- 1- Markov Property

Passing from one sate "x_{n-1}" to the next state "x_n" takes place with respect to the probabilities of "x_n" equal to "k" given "x_{n-1}" equal to "i", shown in equation 5-2 where "M" is the number of states.

$$P(x_n = k|x_{n-1} = i)$$

$$\sum_{K=1}^{M} P\,(x_n = k|x_{n-1} = i) = 1$$

Equation 5- 2- Transition probability between the states.

A hidden Markov model is a kind of Markov chain in which the state is not directly observable, and the observation depends on the state of the system.

HMM has some important attributes. First, there is the transition matrix which stores the transition probability, shown in Equation 5-3. The second characteristic is emission matrix which stores the emission probability. Each element in this matrix shows the probability that the emitted symbol is equal to "i" given the current system is "j" (Equation 5-4).

$$A(i,j) = P(x_n = i | x_{n-1} = j)$$

Equation 5- 3- Transition Matrix

$$B(i,j) = P(y = i | x = j)$$

Equation 5- 4- Emission Matrix

In general, in HMM we want to estimate the emission matrix "B", the transition matrix "A", the probability vector "π" that best match the observed sequence of y_1, …, y_N.

In this laboratory we were asked to work with two different implementations of HMM, Baum-Welch and Viterbi to classify the data. These are already provided by MATALB libraries.

As we learned in the class, the Baum-Welch algorithm takes into consideration forward loop probability and backwards loop probability. Forward loop probability $\alpha_i(n)$ is the probability that the machine is in state "i" at the time "n" emits the observed sequence y_1, …, y_N., with the current parameter θ of the HMM (Equation 5-5).

$$\alpha_i(n) = P(y_1, y_2, \ldots, y_n, x_n = i|\theta), \quad i = 1, \ldots, M, \quad n = 1, \ldots, N$$

Equation 5- 5- Forward loop probability

Furthermore, backwards probability $\beta_i(n)$ is the probability that we observe y_{n+1}, y_{n+2}, …, y_n given that the machine is in state "i" at time n and with the current parameter θ of the HMM (Equation 5-6)

$$\beta_i(n) = P(y_{n+1}, y_{n+2}, \ldots, y_n, x_n = i|\theta), \quad i = 1, \ldots, M, \quad n = 1, \ldots, N-1$$

Equation 5- 6- Backward loop probability

The Viterbi training algorithm is based on the Viterbi algorithm, given the HMM model θ, this algorithm tries to find the sequence of state that maximizes the joint probability. The Viterbi algorithm updates the model θ, using the sequence of state found by the Viterbi algorithm.

5-4-1- HMM Results

Since we fill the transition and emission matrixes with random variables, different random seed has been tested which lead us to have interesting results. By having different random seed, we achieved different results. Therefore, to have a good classification we needed to start with the good random variables in both matrixes. Table 5-1 shows the results for Baum-Welch algorithm for both training set and test set. As it is noticeable from the table, Baum-Welch algorithm classified the healthy people correctly in both train and test set, but it is not able to detect the sick people correctly also, the train results were better than test set results.

Table 5- 1- Baum-Welch results

	Random seed =4444		Random seed =1111		Random seed =1234	
	Train set	**Test Set**	**Train set**	**Test Set**	**Train set**	**Test Set**
No. Healthy	5	5	5	5	5	5
No. Sick	5	5	5	5	5	5
Total No.	10	10	10	10	10	10
No. Healthy Detection	5	5	5	5	5	4
No. Sick Detection	3	2	3	2	4	1
Accuracy	80%	70%	80%	70%	90%	50%

Table 5-2 shows the results for Viterbi model.

Table 5- 2- Viterbi results

	Random seed =4444		Random seed =1111		Random seed =1234	
	Train set	**Test Set**	**Train set**	**Test Set**	**Train set**	**Test Set**
No. Healthy	5	5	5	5	5	5
No. Sick	5	5	5	5	5	5
Total No.	10	10	10	10	10	10
No. Healthy Detection	4	5	4	4	4	2
No. Sick Detection	4	4	4	1	2	0
Accuracy	80%	90%	80%	50%	60%	20%

In general, we can say that Baum-Welch works better than Viterbi algorithm.