ICT for Health
Laboratory # 1
Regression on Parkinson data

Monica Visintin

Politecnico di Torino

October 21st 2022

Table of Contents

Table of Contents

Parkinson's disease [1]

Very short description:

- Patients affected by Parkinson's disease cannot exactly control their **muscles**. In particular they show tremor, they walk with difficulties and, in general, they have problems in starting a movement. Many of them cannot **speak correctly**, since they cannot control the vocal chords and the vocal tract. It has been shown that they overcome the illness if they dance or have an **external clock** that gives the time.
- **Levodopa** is prescribed to the patients, but most of the medicine, which should be absorbed in the intestine, is absorbed by the stomach; as the movements become slower and slower, levodopa stays more and more in the stomach and cannot reach the intestine.
- The beneficial effects of levodopa last for some time, and then a new dose of levodopa should be taken. The neurologist decides when the patient should take levodopa and how much levodopa he/she should take, but it is difficult for the neurologist to **optimize the treatment**, because of the continuous progression of the illness.

Parkinson's disease [2]

- The severity of the illness is measured by neurologists, who judge the patients by asking them to perform many movements (for example tapping the other four fingers with the thumb, or rising from a chair, or walking a short distance, or saying some words) and judging the quality of their life (able to dress? able to prepare his/her own meals?). Adding together the scores gives the final grade, which is called total **UPDRS** (Unified Parkinson's Disease Rating Scale). The visit takes a lot of time, different neurologists may give slightly different scores.
- It would be useful to find an **automatic way** to give the patient an objective score, which can be measured **several times during the day** and help the neurologist to optimize the treatment.

Parkinson's disease [3]

- One possibility is to use parameters of voice to predict the total UPDRS: it is then sufficient to record voice samples (for example using a smartphone), generate these voice parameters (features) and then use a regression technique to predict UPDRS. Unfortunately, Parkinson's disease not always affects voice, and therefore the method can be used only for a subset of patients.
- Goal of the lab is to use **linear regression to predict total UPDRS from a set of voice parameters and other features**.

Table of Contents

Prepare and analyze the data [1]

- Download from (Irvine University, California) `https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring` the Data Folder and Data Set Description. In particular, download files `parkinsons_updrs.data` and `parkinsons_updrs.names` from `https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/telemonitoring/`
  These files are also available in DropBox, folder `/visintin/laboratories/lab1_Parkinsons`.
  The data were obtained by some researchers who evaluated in the same day total UPDRS and motor UPDRS of some patients and recorded the speech of the patients to measure voice parameters. The patients were analyzed several times in 6 months and several records exist for each patient, at different times. The outcome is a matrix with many rows (one for each measurement) and many columns (one for each feature).

Prepare and analyze the data [2]

- File `parkinsons_updrs.data` stores F columns and N rows; the columns are separated by commas (csv file). Change the extension of the file from `.data` to `.csv`, so that you can have a look at the file using Excel or LibreOfficeCalc. File `parkinsons_updrs.csv` is already available in DropBox.
- A useful Python library that can be used to analyze data is `Pandas`: if you have not installed it yet, download it.
- Open Spyder3 or Pycharm or Jupyter or Colab. Start a new script in which you import pandas, matplotlib.pyplot, NumPy:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
```

Prepare and analyze the data [3]

- Read the Parkinson's data file using Pandas function `pd.read_csv(''filename'')` .

```
4 x=pd.read_csv("parkinsons_updrs.csv")
```

This line works if your script and file `parkinsons_updrs.csv` are both in your working folder.

- Search the web for the usage of Pandas. The main things you have to know is that `x` is a **DataFrame**, and that many attributes and methods exist for DataFrames (see `https://pandas.pydata.org/pandas-docs/stable/api.html#id2`).

Prepare and analyze the data [4]

- Examples of methods associated with DataFrames:

```
features=list(x.columns)}#list with the names of the features
x.info()#gives you information about the data (number of valid values, type
x.describe().T#descr. of dataset (min, max, mean, etc of each feat.)
x.plot.hist(bins=50)#plots the histograms of all the features
x.plot.scatter('a','b')#plots the scatter plot, i.e. x.b versus x.a
x.cov()}#gives the covariance matrix for the features in the columns
x.values()#gives the NumPy Ndarray with the data
```

Prepare and analyze the data [5]

- Start checking the data (mandatory step each time you work with a new dataset):
  1. Write in your code

```
x.describe().T
x.info()
```

and look at the printed values. Check that there are no major problems with the data (no missing values, no out-of-scale values, etc). `x.info()` gives the following output

```
subject#          5875 non-null int64
age               5875 non-null int64
sex               5875 non-null int64
test_time         5875 non-null float64
motor_UPDRS       5875 non-null float64
total_UPDRS       5875 non-null float64
Jitter(%)         5875 non-null float64
Jitter(Abs)       5875 non-null float64
Jitter:RAP        5875 non-null float64
Jitter:PPQ5       5875 non-null float64
Jitter:DDP        5875 non-null float64
```

Prepare and analyze the data [6]

```
12 Shimmer              5875 non-null float64
13 Shimmer(dB)          5875 non-null float64
14 Shimmer:APQ3         5875 non-null float64
15 Shimmer:APQ5         5875 non-null float64
16 Shimmer:APQ11        5875 non-null float64
17 Shimmer:DDA          5875 non-null float64
18 NHR                  5875 non-null float64
19 HNR                  5875 non-null float64
20 RPDE                 5875 non-null float64
21 DFA                  5875 non-null float64
22 PPE                  5875 non-null float64
```

which means that there are 5875 row with no missing values and the read values are either integer or float numbers.

Prepare and analyze the data [7]

2. Check the names/meanings of the available features:

```
5 features=list(x.columns)
6 print(features)
```

The list of features is 'subject#', 'age', 'sex', 'test_time', 'motor_UPDRS', 'total_UPDRS', 'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP', 'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11', 'Shimmer:DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE'.
Note that you can access to the values of feature 'age' by writing `x.age` (you get a Pandas series, one column).

- Feature 'total_UPDRS' is the regressand.
- Using line

```
7 subj=pd.unique(x['subject#'])# existing values of patient ID
8 print("The number of distinct patients in the dataset is ",
9         len(subj))
```

Prepare and analyze the data [8]

we check that there are 42 patients with distinct IDs.

- Features 'age' and 'sex' are obvious, they will be regressors.
- Feature 'test_time' is a float where the integer part is the day (from the beginning of the measurement period of the patient) and the decimal part is the hour. A modified version of 'test_time' will be a regressor.
- **Jitter** is the variation of the fundamental **frequency** (or, conversely, its period) in signals that should be periodic but are not (it is impossible that the frequency of a sinusoidal signal generated by an electronic equipment never changes; it is impossible that a human generated vocal signal like 'a' is perfectly periodic). **Shimmer** is the variation of **amplitude** in signals that should be periodic but are not. NHR is the noise to harmonics ratio; HNR is the harmonics to noise ratio. RPDE is Recurrence Period Density Entropy, DFA is the Detrended Fluctuation Analysis, PPE is Perceived Vocal Effort. All these features/parameters are related to voice and are automatically evaluated by specific software that uses a voice signal as input.
- Other features could be extracted from voice signals, for example the maximum value, the minimum value, the maximum of the absolute value of the FFT (Fast Fourier Transform) of the signal. The process of **extracting relevant features** is complex. In this lab we use the features available in the dataset.

Prepare and analyze the data [9]

3. We want to have in one day only the average values of voice parameters (UPDRS in only measured once in a day). Pandas method `groupby` allows to do this quickly. Write in your script:

```
X=pd.DataFrame()
for k in subj:
    xk=x[x['subject#']==k]# data of user k
    xk1=xk.copy()# we modify the values of xk (next lines);
    # a warning would be issued if we did not make a copy
    xk1.test_time=xk1.test_time.astype(int)# remove decimal values
    xk1['g']=xk1['test_time']# add a new feature
    v=xk1.groupby('g').mean()# group according to the new feature
    # which is removed
    X=pd.concat([X,v],axis=0,ignore_index=True)# append new data to X
features=list(x.columns)
print("The dataset shape after the mean is ",X.shape)
print("The features of the dataset are ",len(features))
print(features)
Np,Nc=X.shape# Np = number of rows/ptients
# Nc=number Nf of regressors + 1 (regressand total UPDRS is included)
```

After this, the shape of the dataframe is (990,22).

Prepare and analyze the data [10]

4. Let us check if features are correlated. Method `X.cov()` gives the covariance of the dataset, but unfortunately feature 'test_time' has a large variance, that makes the other covariance values too small to be seen in an image. Therefore, it is necessary to first normalize the data, then use the DataFrame method `cov` to evaluate the covariance matrix and plot it using Matplotlib. In practice, instead of showing the covariance

$$C[i,j] = \mathbb{E}\{(X_i - \mu_i)(X_k - \mu_k)\}$$

of the random variables X_i and X_k, we want to see the correlation coefficient

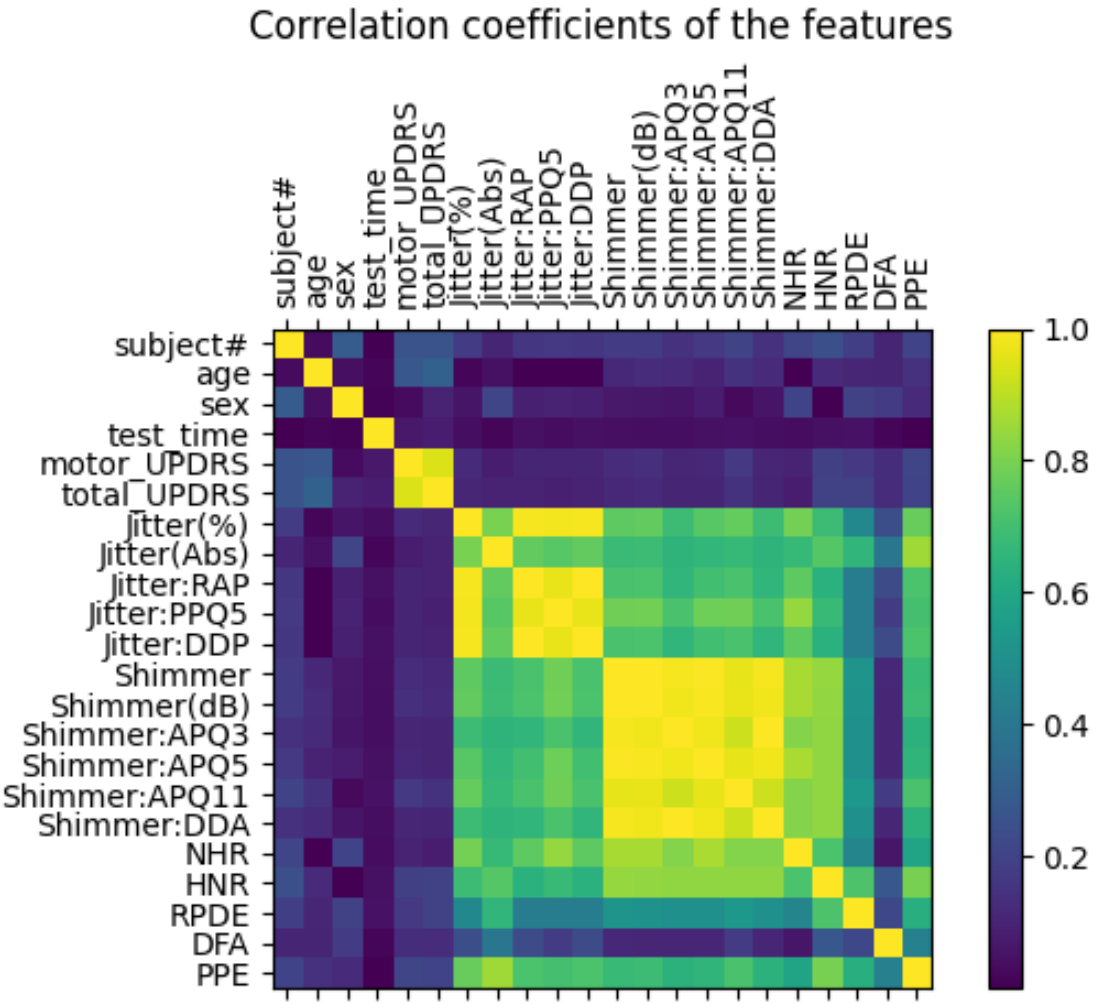
$$\rho[i,j] = \frac{\mathbb{E}\{(X_i - \mu_i)(X_k - \mu_k)\}}{\sigma_i \sigma_k}$$

($\mu_i, \mu_k, \sigma_i, \sigma_k$ are the means and standard deviations of the random variables X_i and X_k, respectively).
Note that we are just observing the data, we are not yet performing regression and therefore we use the entire dataset.
Remember that the correlation coefficient can only take values in the range $[-1, 1]$.

Prepare and analyze the data [11]

```
25 Xnorm=(X-X.mean())/X.std()#normalize the entire dataset
26 c=Xnorm.cov()#measure the covariance
27 plt.figure()
28 plt.matshow(np.abs(c.values),fignum=0)
29 plt.xticks(np.arange(len(features)), features, rotation=90)
30 plt.yticks(np.arange(len(features)), features, rotation=0)
31 plt.colorbar()
32 plt.title('Correlation coefficients of the features')
33 plt.tight_layout()
34 plt.savefig('./corr_coeff.png')# save the figure
35 plt.show()
```

Prepare and analyze the data [12]

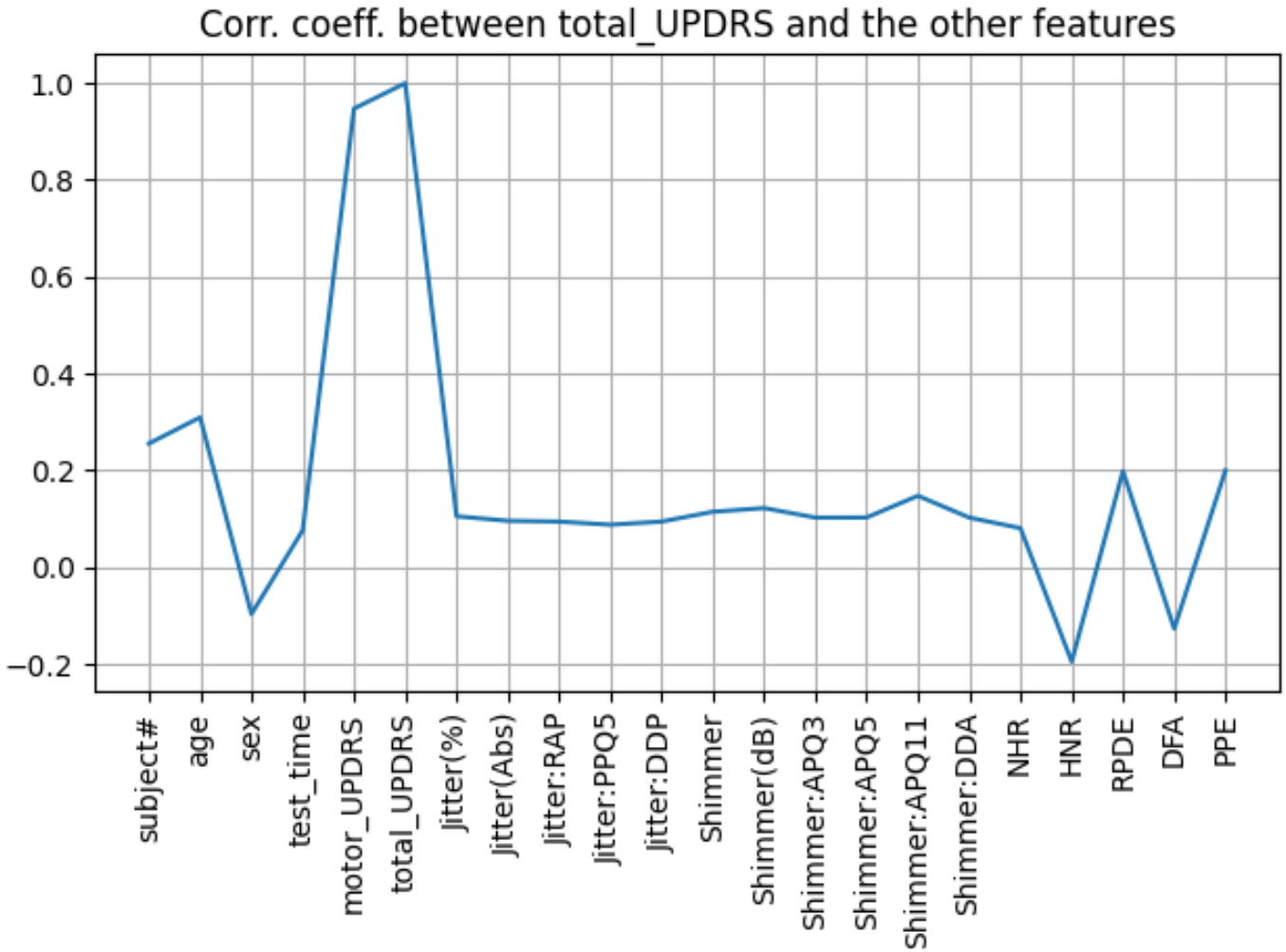


Prepare and analyze the data [13]

5. Note that motor and total UPDRS are highly correlated (obviously), but also the various jitter parameters are correlated among themselves, and the same occurs for shimmer parameters. This might give rise to **collinearity** or multicollinearity: one feature among the regressors can be linearly derived from other regressors, which means that many different vectors $\mathbf{w}$ exist that solve the problem with the same value of the objective function. In such cases it might be convenient to **remove all but one of the linearly dependent features**, selecting the one that has the highest correlation coefficient with the regressand (total UPDRS in our case). However we will first keep all the features to see what happens, and then we will remove some of them.
6. Look also at the values in DataFrame `c` related to total UPDRS:

```
36 plt.figure()
37 c.total_UPDRS.plot()
38 plt.grid()
39 plt.xticks(np.arange(len(features)), features, rotation=90)
40 plt.title('corr. coeff. among total UPDRS and the other features')
41 plt.tight_layout()
42 plt.show()
```

Prepare and analyze the data [14]



Prepare and analyze the data [15]

7. Clearly motor UPDRS is correlated to total UPDRS (it is a part of it and we know), whereas the correlation between total UPDRS and voice features is not so large.
8. The subject ID is correlated to total UPDRS, but it is not correct to use it as regressor: a new patient will have a completely different ID, we cannot allow that his/her regressed total UPDRS depend on the ID. We will later drop the subject ID from the list of regressors.
9. This initial investigation takes time, it is annoying, but it is **mandatory**, otherwise big errors are possible. You must be aware of the meaning of the data you are processing. It is not a matter of blindly running an algorithm, people health is at stake.

Prepare and analyze the data [16]

- Prepare a new DataFrame in which you **randomly permute (shuffle)** the rows of the original DataFrame. This operation avoids that the data of only the first patients appear in the training dataset. The rationale behind this operation is that we pretend that all the measurements are related to different patients at different stages in the illness evolution. We then take the first rows of the DataFrame to train/validate the regression model and the remaining rows to test the performance of the found model. Shuffling is performed setting the seed, to have reproducibility of the script.

Prepare and analyze the data [17]

First solution to shuffle the data:

```
43 np.random.seed(101) # set the seed for random shuffling
44 indexsh=np.arange(Np)
45 np.random.shuffle(indexsh)
46 Xsh=X.copy(deep=True)
47 Xsh=Xsh.set_axis(indexsh,axis=0,inplace=False)
48 Xsh=Xsh.sort_index(axis=0)
```

Second solution to shuffle the data:

```
43 Xsh=X.sample(fract=1,replace=False,random_state=101,axis=0,
44     ignore_index=True)
```

- The regressand will be **total UPDRS**, whereas all the other features (including motor UPDRS and excluding subject ID) will be regressors.

Perform regression [1]

- It is now time to start generating the regression model. We assume that random variable total_UPDRS linearly depends on the random variables sex, age, motor_UPDRS, shimmer, jitter, etc:

$$Y = w_1 X_1 + w_2 X_2 + \cdots + w_{N_f} X_{N_f} + \nu$$

where $Y, X_1, \ldots, X_{N_f}, \nu$ are all random variables and $w_1, \ldots, w_{N_f}$ are the weights to be found.

Perform regression [2]

- In the previous model we assume that all the random variables have zero mean. A more complete model is

$$Y = w_1 X_1 + w_2 X_2 + \cdots + w_{N_f} X_{N_f} + C + \nu$$

but it is convenient to work with features with zero-mean and variance one which typically reduces numerical problems and speeds up algorithm convergence. Therefore we first perform **normalization** of the data, by removing the mean and divide by the standard deviation each random variable.
In order to be consistent with our scenario, we must consider that **the feature means and standard deviations can only be measured on the training dataset**: when we train the regression model we do not know the parameters of **future patients** that will appear in the test dataset, do we?

Perform regression [3]

- We will use the first 50% of the rows (495) of the shuffled matrix as **training** points (define the new DataFrame as `X_tr`), and the remaining 50% of the rows (495) for **testing** (define the new DataFrame as `X_te`). This is accomplished with the instructions

```
45 Ntr=int(Np*0.5)  # number of training points
46 Nte=Np-Ntr    # number of test points
47 X_tr=Xsh[0:Ntr]# dataframe that contains only the training data
48 mm=X_tr.mean()# mean (series) of the training data
49 ss=X_tr.std()# standard deviation (series) of the training data
50 my=mm['total_UPDRS']# mean of total UPDRS
51 sy=ss['total_UPDRS']# st.dev of total UPDRS
```

A series is substantially a DataFrame with just one column (not exact, just to give an idea).

Perform regression [4]

- Now we can normalize the data and split them into training, validation, test subsets. At the moment only training and test subsets are needed (LLS solution). Remember that validation is a subset of training dataset.

```
52 Xsh_norm=(Xsh-mm)/ss# normalized data
53 ysh_norm=Xsh_norm['total_UPDRS']# regressand
54 Xsh_norm=Xsh_norm.drop(['total_UPDRS','subject#'],axis=1)# regressors
55
56 X_tr_norm=Xsh_norm[0:Ntr]# training regressors
57 X_te_norm=Xsh_norm[Ntr:]# test regressors
58 y_tr_norm=ysh_norm[0:Ntr]# training regressand
59 y_te_norm=ysh_norm[Ntr:]# test regressand
```

Perform regression [5]

- Regression procedure:
  1. DataFrames `X_tr_norm` and `Y_tr_norm` can be directly used to find (no need to move to Numpy NDarrays):

$$\hat{\mathbf{w}} = \arg\min ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2 / N$$

(`X_tr_norm`=$\mathbf{X}$, `Y_tr_norm`=$\mathbf{y}$, `Ntr`=N). With LLS:

```
60 w_hat=np.linalg.inv(X_tr_norm.T@X_tr_norm)@(X_tr_norm.T@y_tr_norm)
```

however slicing with Pandas is more complex than in Numpy, so we suggest to use Numpy:

Perform regression [6]

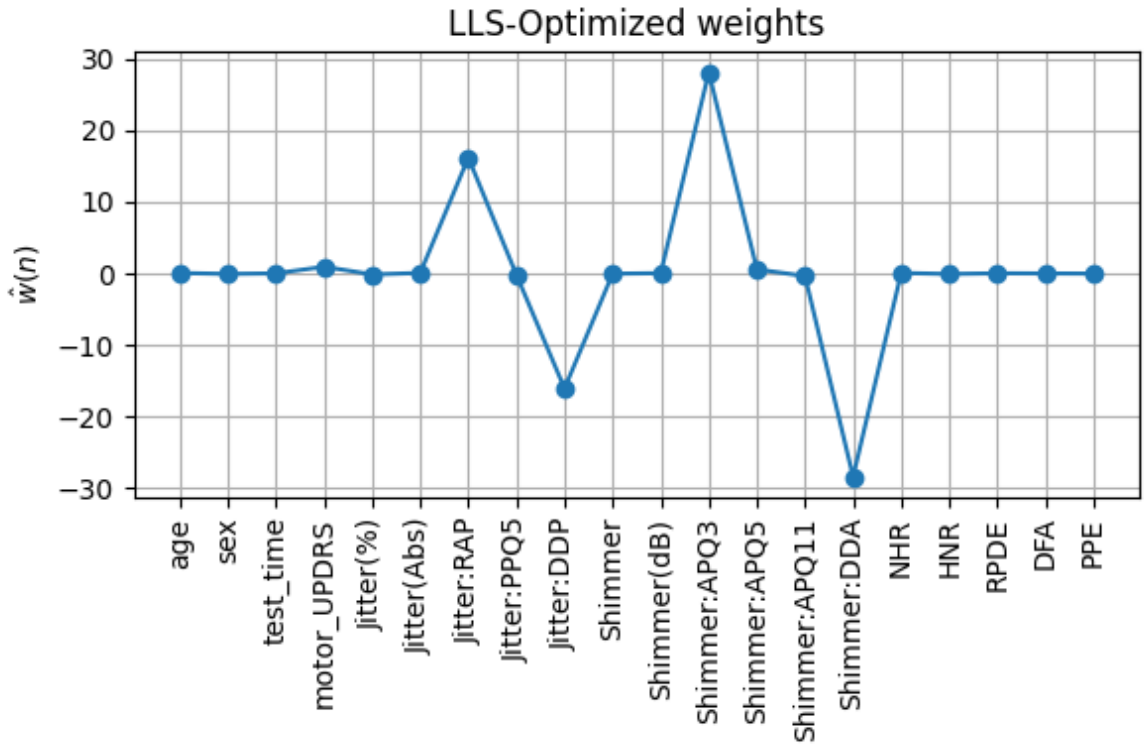
```
52 Xsh_norm=(Xsh-mm)/ss# normalized data
53 ysh_norm=Xsh_norm['total_UPDRS']# regressand
54 Xsh_norm=Xsh_norm.drop(['total_UPDRS','subject#'],axis=1)# regressors
55 Xsh_norm=Xsh_norm.values
56 ysh_norm=ysh_norm.values
57 X_tr_norm=Xsh_norm[0:Ntr]# training regressors
58 X_te_norm=Xsh_norm[Ntr:]# test regressors
59 y_tr_norm=ysh_norm[0:Ntr]# training regressand
60 y_te_norm=ysh_norm[Ntr:]# test regressand
61 w_hat=np.linalg.inv(X_tr_norm.T@X_tr_norm)@(X_tr_norm.T@y_tr_norm)
```

Perform regression [7]

2. Once you find $\hat{\mathbf{w}}$, you must plot it in order to find potential problems

```
62 regressors=list(X_tr_norm.columns)
63 Nf=len(w_hat)
64 nn=np.arange(Nf)
65 plt.figure(figsize=(6,4))
66 plt.plot(nn,w_hat,'-o')
67 ticks=nn
68 plt.xticks(ticks, regressors, rotation=90)
69 plt.ylabel(r'$\^w(n)$')
70 plt.title('LLS-Optimized weights')
71 plt.grid()
72 plt.tight_layout()
73 plt.savefig('./LLS-what.png')
74 plt.show()
```

Perform regression [8]



The effect of **collinearity** can be seen since, for example, the weights associated with 'Shimmer:APQ3' and 'Shimmer:DDA' are very large and with opposite signs. For the moment we keep all the regressors and we finish the analysis, you must drop the two features Jitter:DDP and Shimmer:DDA in your version of the script.

Perform regression [9]

3. Then, having found $\hat{\mathbf{w}}$, the test dataset will be used to evaluate

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$$

(`X_te_norm`=$\mathbf{X}$). Goodness of $\hat{\mathbf{w}}$ will be measured by comparing $\hat{\mathbf{y}}$ and `Y_te_norm`. However, we are interested also in the training dataset performance to find out a possible overfitting phenomenon:

```
75 y_hat_te_norm=X_te_norm@w_hat
76 y_hat_tr_norm=X_tr_norm@w_hat
```

Perform regression [10]

4. However, note that `y_te_norm` and `y_hat_te_norm` are normalized, and therefore the value of MSE (mean Square Error) does not say much to a medical doctor (what is the unit of measurement?), and it is necessary to de-normalize $\hat{\mathbf{y}}$:

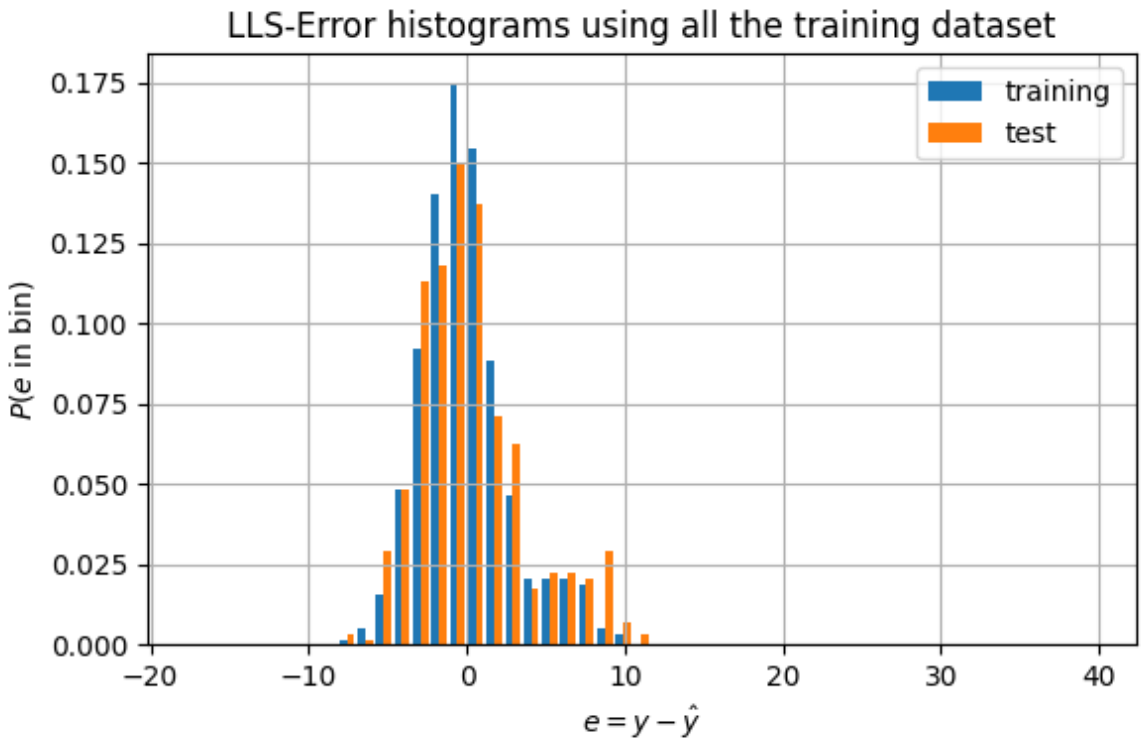
```
77 y_hat_te=y_hat_te_norm*sy+my
78 y_te=y_te_norm*sy+my
79 y_hat_tr=y_hat_tr_norm*sy+my
80 y_tr=y_tr_norm*sy+my
```

Performance figures [1]

1. You must check if the regression error shows peculiar trends, which might reveal an error in the script. A histogram of the error $Y - \hat{Y}$ is useful.

```
81 E_tr=(y_tr-y_hat_tr)# training
82 E_te=(y_te-y_hat_te)# test
83 e=[E_tr,E_te]
84 plt.figure(figsize=(6,4))
85 plt.hist(e,bins=50,density=True, histtype='bar',
86 label=['training','test'])
87 plt.xlabel(r'$e=y-\^y$')
88 plt.ylabel(r'$P(e$ in bin$)$')
89 plt.legend()
90 plt.grid()
91 plt.title('LLS-Error histograms')
92 plt.tight_layout()
93 plt.savefig('./LLS-hist.png')
94 plt.show()
```

Performance figures [2]



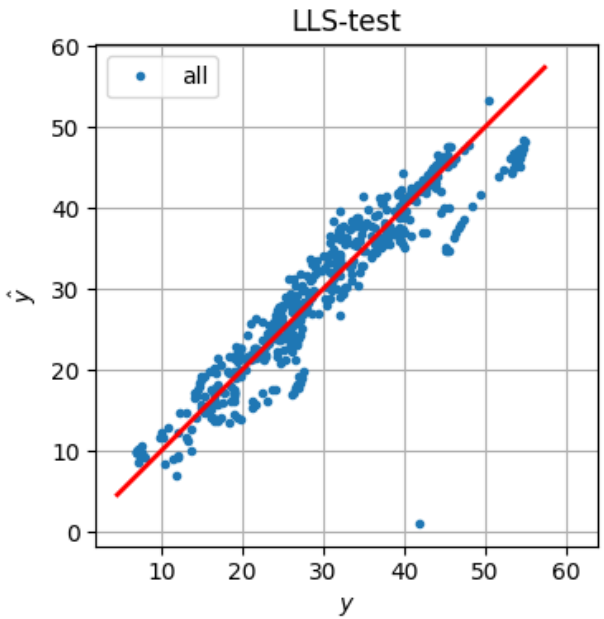
Clearly, the error does not have a Gaussian pdf, it is a mixture of two Gaussian pdfs, and there is not much difference in the training and test subsets (which means that there is no **overfitting**).

Performance figures [3]

2. You must compare the true and the regressed value of y (regression line). **Note: when you plot a versus b, a is on the y-axis and b is on the x-axis.**

```
95  y_hat_te=(X_te_norm@w_hat)*sy+my
96  y_te=y_te_norm*sy+my
97  plt.figure(figsize=(6,4))
98  plt.plot(y_te,y_hat_te,'.')
99  v=plt.axis()
100 plt.plot([v[0],v[1]],[v[0],v[1]],'r',linewidth=2)
101 plt.xlabel(r'$y$')
102 plt.ylabel(r'$\^y$')
103 plt.grid()
104 plt.title('LLS-test')
105 plt.tight_layout()
106 plt.savefig('./LLS-yhat_vs_y.png')
107 plt.show()
```

Performance figures [4]



Note that the estimated values $\hat{y}$ are close to the true values y apart from some cases in which y is very large and $\hat{y}$ takes smaller values with an error around 7-8 UPDRS points (which justifies the second Gaussian-like part of the histogram for error values around 8). Also there is a point with $y = 42$ and $\hat{y}$ close to zero.

Performance figures [5]

3. Other important parameters are min, max, mean, standard deviation, mean square value of the error in each of the subsets, R^2 (coefficient of determination), correlation coefficient:

```
108 E_tr_min=E_tr.min()
109 E_tr_max=E_tr.max()
110 E_tr_mu=E_tr.mean()
111 E_tr_sig=E_tr.std()
112 E_tr_MSE=np.mean(E_tr**2)
113 R2_tr=1-E_tr_MSE/(np.std(y_tr**2))
114 c_tr=np.mean((y_tr-y_tr.mean())*(y_hat_tr-y_hat_tr.mean()))
115 E_te_min=E_te.min()
116 E_te_max=E_te.max()
117 E_te_mu=E_te.mean()
118 E_te_sig=E_te.std()
119 E_te_MSE=np.mean(E_te**2)
120 R2_te=1-E_te_MSE/(np.std(y_te)**2)
121 c_te=np.mean((y_te-y_te.mean())*(y_hat_te-y_hat_te.mean()))
```

Performance figures [6]

To show the results in a better form, a DataFrame can be generated and printed:

```
122 rows=['Training','Test']
123 cols=['min','max','mean','std','MSE','R^2','corr_coeff']
124 p=np.array([
125                 [E_tr_min,E_tr_max,E_tr_mu,E_tr_sig,E_tr_MSE,R2_tr,c_tr],
126                 [E_te_min,E_te_max,E_te_mu,E_te_sig,E_te_MSE,R2_te,c_te]
127                 ])
128 results=pd.DataFrame(p,columns=cols,index=rows)
129 print(results)
```

The printed output is

```
1                min       max          mean    std     MSE    R^2  corr_coeff
2 Training  −7.504    10.469  −8.198e−14  2.886   8.326  0.922       0.960
3 test      −7.539    40.810   3.786e−01  3.905  15.390  0.873      0.9351
```

Conclusions that can be drawn looking at the above results are:

Performance figures [7]

- The error mean in the training subset is zero (as it should be); the error mean is slightly positive in the other two subsets because the mean of total UPDRS were evaluated using only the training subset, and it is therefore possible that total UPDRS means in the test subset are not exactly zero, after normalization.
- The error for the test dataset has one "outlier": the regressed value is much lower than the true value and the maximum error is 40.8. This only large error accounts for the higher error standard deviation and mean square value for the test dataset with respect to the training dataset.
- The coefficient of determination R^2 is close to around 0.9, as the correlation coefficient, which means that the regression is pretty good, in spite of the error at the single point.

Performance figures [8]

- Since the error standard deviation is around 3-4 points, this means that most of the times the regression error is around 6-8 points (see also the histogram), which might still be accepted by a medical doctor. Regression is not very precise, but having an error of 6 points when total_UPDRS is equal to 50 points is still reasonable. Note that the regressand (UPDRS) has standard deviation equal to 10.34 points and mean value 28.50 points, therefore without regression we should predict a UPDRS value of 28.50 and we would have an error standard deviation equal to 10.34 points, which is about 2.5 times what the regression model provides.
- It has to be noticed that the model was obtained in the presence of collinearity and the performance might improve by removing it. On the contrary, total UPDRS was not regressed from voice parameters only, but also from motor UPDRS: if it is dropped then the performance is much worse.

Table of Contents

What you have to do [1]

- Drop features Jitter:DDP and Shimmer:DDA.
- Use your ID/matricola number as random seed.
- Add regression (exactly the same steps shown for LLS) using steepest descent, with a suitable stopping condition (you are responsible of your decisions). Try and use classes and inherited methods or methods (for example to plot histograms, to plot $\hat{y}$ versus y, to generate the DataFrame with the results, etc), exploit the code you already wrote. Compare results obtained with steepest descent with those obtained using LLS (same input dataset, i.e. drop Jitter:DDP and Shimmer:DDA also for LLS).

What you have to do [2]

- We have seen that linear regression is somehow similar to interpolation, Taylor series etc. We can improve the results by using a local linear regression model: given a test point, instead of finding $\hat{\mathbf{w}}$ from all the points in the training dataset, only the N-th closest point are used. Stated in another way: for each of the normalized test points $\mathbf{x}$, find the N points in the normalized training dataset $\mathbf{X}_{tr}$ that are closer to $\mathbf{x}$; generate a new training dataset with matrix $\mathbf{X}_{tr}(\mathbf{x})$ and vector $\mathbf{y}_{tr}(\mathbf{x})$, both with N rows only; find $\hat{\mathbf{w}}(\mathbf{x})$ that minimizes $\|\mathbf{X}_{tr}(\mathbf{x})\mathbf{w} - \mathbf{y}_{tr}(\mathbf{x})\|^2$; then find the normalized regressed value $\hat{y} = \mathbf{x}^T\hat{\mathbf{w}}(\mathbf{x})$. Note that each point $\mathbf{x}$ in the test dataset will have a different weight vector $\hat{\mathbf{w}}(\mathbf{x})$ (and you cannot plot 495 weight vectors). Use steepest descent to find $\hat{\mathbf{w}}(\mathbf{x})$. As for the value of N, start with 10, but check some values to see if this parameter influences the final results.

What you have to do [3]

- Generate at least the following plots for regression of total UPDRS for each of the methods:
  1. The **estimated regressand versus the true regressand** for the test dataset (note: the de-normalized values, not the normalized values).
  2. The **histograms** (de-normalized), of the estimation error for training validation and test datasets.
- Fill in a **table** (DataFrame) with the values of the measured min, max, mean, standard deviation, mean square value for the regression errors (de-normalized) and R^2 and correlation coefficient for the test dataset, and draw your conclusions. The table must include the results obtained with normal linear regression and local linear regression, to ease comparisons.
- Once your script runs with the seed equal to your PoliTO ID number, run it at least 20 times with 20 different seeds and average the results. The report must include the table valid for the seed equal to your PoliTO ID number and the table with the averaged values over the 20 runs.

What you have to do [4]

- VERY IMPORTANT: in the applications analyzed in this course **execution speed** is not an issue (we are not dealing with big data); algorithm **complexity** is not an issue (think of the cost of a "simple" ultrasound machine, around 60-100 kEuros, and the cost of a "good" server with CUDAs etc, let's say 5-10 KEuros). If you need more CPUs, you buy them. The true issue is **reliability** and goodness of the result. As already remarked, patient's health is at stake.
- If an algorithm gives you $R^2 = 0.986$ and another one gives you $R^2 = 0.987$, the two algorithms are equivalent, you cannot say that the first is better than the second because R^2 has a lower third decimal digit, such a difference is irrelevant from an engineering point of view.
- **Report due for November 17th 11.59 PM, maximum 6 pages**.
- The report must be written as a research paper, all the details must be given so that the experiment can be exactly reproduced by another researcher on the other side of the world in 2060 (when you will not be available to give the missing details).

What you have to do [5]

- Since the application is in the health field, a particular emphasis must be given to the medical point of view, both in the introduction and the conclusions.
- A lecture will be recorded to show you how to write and upload the report.
- Remember that you pay university fees to learn. If you do not work you do not learn, you are wasting your money and your time.