ICT for Health
Laboratory # 1
Regression on Parkinson data

Monica Visintin

Politecnico di Torino

2024/25

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.
- 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]

- From DropBox download file `parkinsons_updrs_av.csv`. This file was obtained by processing an original dataset available at (Irvine University, California) `https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring`
- The features available in the dataset are those described at `https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring`
- 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]

- A useful Python library that can be used to analyze data is `Pandas`: if you have not installed it yet, download it.
- Open Spyder or Pycharm or Jupyter or Colab. Start a new script in which you import pandas, matplotlib.pyplot, NumPy:

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Prepare and analyze the data [3]

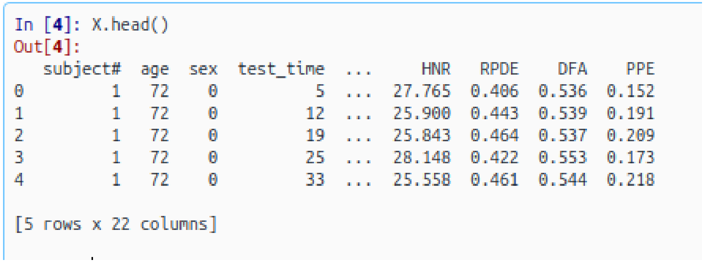
- Read the Parkinson's data file using Pandas function `pd.read_csv(‘‘filename’’)` .

```
X=pd.read_csv("parkinsons_updrs_av.csv")
```

  This line works if your script and file `parkinsons_updrs_av.csv` are both in your working folder. If your working folder is different, you must specify the entire path of `parkinsons_updrs_av.csv` as argument of `pd.read_csv`.
- Search the web or ask ChatGPT 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]

A dataframe is essentially an “organized” table/matrix. Command `X.head()` shows the first lines of dataframe `X`:



```
In [4]: X.head()
Out[4]:
   subject#  age  sex  test_time  ...     HNR   RPDE    DFA    PPE
0         1   72    0          5  ...  27.765  0.406  0.536  0.152
1         1   72    0         12  ...  25.900  0.443  0.539  0.191
2         1   72    0         19  ...  25.843  0.464  0.537  0.209
3         1   72    0         25  ...  28.148  0.422  0.553  0.173
4         1   72    0         33  ...  25.558  0.461  0.544  0.218

[5 rows x 22 columns]
```

Each column has the corresponding column name, all the rows have an index (integer from 0 onwards). You do not have to remember that feature age is the third column, you directly access to `X.age` or `X['age']`.

Prepare and analyze the data [5]

- 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 [6]

- 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).

Prepare and analyze the data [7]

`X.info()` gives the following output

```
0   subject#        990 non-null    int64
1   age             990 non-null    int64
2   sex             990 non-null    int64
3   test_time       990 non-null    int64
4   motor_UPDRS     990 non-null    float64
5   total_UPDRS     990 non-null    float64
6   Jitter(%)       990 non-null    float64
7   Jitter(Abs)     990 non-null    float64
8   Jitter:RAP      990 non-null    float64
9   Jitter:PPQ5     990 non-null    float64
10  Jitter:DDP      990 non-null    float64
11  Shimmer         990 non-null    float64
12  Shimmer(dB)     990 non-null    float64
13  Shimmer:APQ3    990 non-null    float64
14  Shimmer:APQ5    990 non-null    float64
15  Shimmer:APQ11   990 non-null    float64
16  Shimmer:DDA     990 non-null    float64
17  NHR             990 non-null    float64
18  HNR             990 non-null    float64
19  RPDE            990 non-null    float64
20  DFA             990 non-null    float64
```

Prepare and analyze the data [8]

```
 21   PPE                  990  non-null     float64
dtypes:  float64(18),  int64(4)
```

which means that there are 990 rows with no missing values and the read values are float numbers.

Prepare and analyze the data [9]

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

```
features=list(x.columns)
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'.
There are 22 features.
Note again that you can access the values of feature 'age' by writing `X.age` or `X.'age'` (you get a Pandas "series", i.e. one column).

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

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

Prepare and analyze the data [10]

we check that there are 42 patients with distinct IDs.

- Features 'age' and 'sex' are obvious, they will be regressors.
- **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 [11]

3. 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

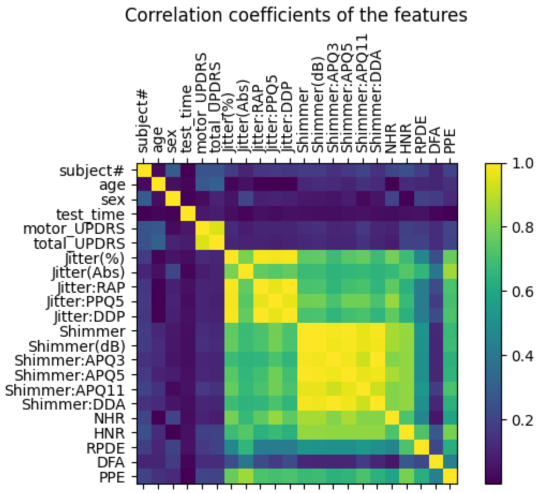
$$\rho[i,j] = \frac{\mathbb{E}\{(X_i - \mu_i)(X_k - \mu_k)\}}{\sigma_i \sigma_k}$$

($\mu_i, \mu_k, \sigma_i, \sigma_k$ 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 [12]

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

Prepare and analyze the data [13]

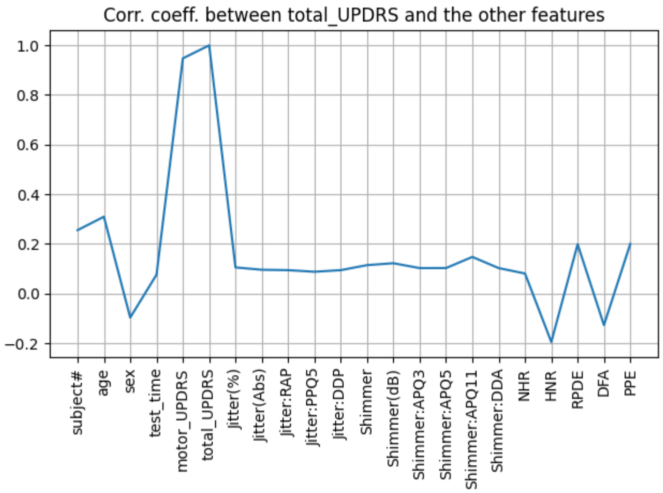


Prepare and analyze the data [14]

4. 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.
5. Look also at the values in DataFrame `c` (the matrix with the correlation coefficients) related to total UPDRS:

```python
plt.figure()
c.total_UPDRS.plot()
plt.grid()
plt.xticks(np.arange(len(features)), features, rotation=90)
plt.title('corr. coeff. among total UPDRS and the other features')
plt.tight_layout()
plt.show()
```

Prepare and analyze the data [15]



6. 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.

Prepare and analyze the data [16]

7. The subject ID is correlated to total UPDRS, but it is not logically 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 this (random) ID. We will later drop the subject ID from the list of regressors.
8. 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 [17]

- 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. Note that results might change by changing the seed

Prepare and analyze the data [18]

First solution to shuffle the data:

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

Second solution to shuffle the data:

```
Xsh=X.sample(fract=1,replace=False,random_state=101,axis=0,
    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 (once you remove the mean from all the random variables, including the regressand, the model has obviously intercept equal to zero). Therefore we first perform **normalization (or standardization)** of the data, by removing the mean and divide by the standard deviation each random variable:

$$X_{norm} = \frac{X - \mu_X}{\sigma_X}; \quad (X = \sigma_X X_{norm} + \mu_X)$$

In order to be consistent with our scenario, **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 lines

```
Ntr=int(Np*0.5)  # number of training points
Nte=Np-Ntr   # number of test points
X_tr=Xsh[0:Ntr]# dataframe that contains only the training data
mm=X_tr.mean()# mean (series) of the training data
ss=X_tr.std()# standard deviation (series) of the training data
my=mm['total_UPDRS']# mean of total UPDRS
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 the training dataset.

```python
Xsh_norm=(Xsh-mm)/ss# normalized data
ysh_norm=Xsh_norm['total_UPDRS']# regressand
Xsh_norm=Xsh_norm.drop(['total_UPDRS','subject#'],axis=1)# regressors

X_tr_norm=Xsh_norm[0:Ntr]# training regressors
X_te_norm=Xsh_norm[Ntr:]# test regressors
y_tr_norm=ysh_norm[0:Ntr]# training regressand
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):
$$\hat{\mathbf{w}} = \text{arg min } \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2/N$$
(`X_tr_norm`=$\mathbf{X}$, `Y_tr_norm`=$\mathbf{y}$, `Ntr`=N). With LLS:

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

Perform regression [6]

However slicing (selection of subsets of rows/columns) with Pandas is sometimes more complex than in Numpy (for which slicing is obvious), so we suggest to use Numpy, as follows:

```
Xsh_norm=(Xsh-mm)/ss# normalized data
ysh_norm=Xsh_norm['total_UPDRS']# regressand
Xsh_norm=Xsh_norm.drop(['total_UPDRS','subject#'],axis=1)# regressors
Xsh_norm=Xsh_norm.values # Xsh_norm is now a Numpy NDarray (matrix)
ysh_norm=ysh_norm.values # ysh_norm is now a Numpy NDarray (vector)
X_tr_norm=Xsh_norm[0:Ntr]# training regressors
X_te_norm=Xsh_norm[Ntr:]# test regressors
y_tr_norm=ysh_norm[0:Ntr]# training regressand
y_te_norm=ysh_norm[Ntr:]# test regressand
w_hat=np.linalg.inv(X_tr_norm.T@X_tr_norm)@(X_tr_norm.T@y_tr_norm)
```

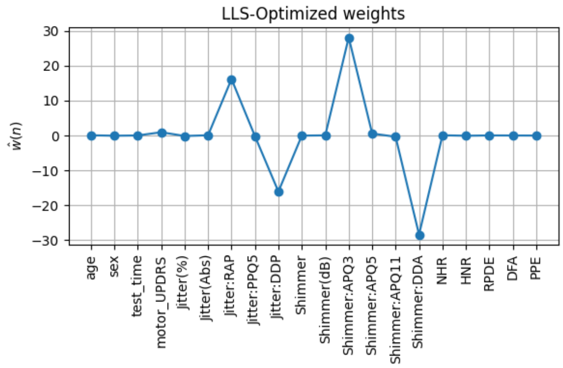
Of course you can use the classes you wrote during the first laboratory to perform the regression.

Perform regression [7]

2. Once you find $\hat{\mathbf{w}}$, you must plot it in order to find potential problems (again, you can use the method we wrote in laboratory 0):

```python
regressors=list(X_tr_norm.columns)
Nf=len(w_hat)
nn=np.arange(Nf)
plt.figure(figsize=(6,4))
plt.plot(nn,w_hat,'-o')
ticks=nn
plt.xticks(ticks, regressors, rotation=90)
plt.ylabel(r'$\^w(n)$')
plt.title('LLS-Optimized weights')
plt.grid()
plt.tight_layout()
plt.savefig('./LLS-what.png')
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 (the same occurs with 'Jitter:RAP' and 'Jitter:DDP'). For the moment we keep all the regressors and we finish the analysis, but you must drop the two features Jitter:DDP and Shimmer:DDA in your final 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:

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

If the mean square error for the training dataset is much smaller than that for the test dataset, then overfitting has occurred.

Perform regression [10]

4. 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 of the normalized mean square error?), and it is necessary to first de-normalize $\hat{\mathbf{y}}$ to get a meaningful mean square error:

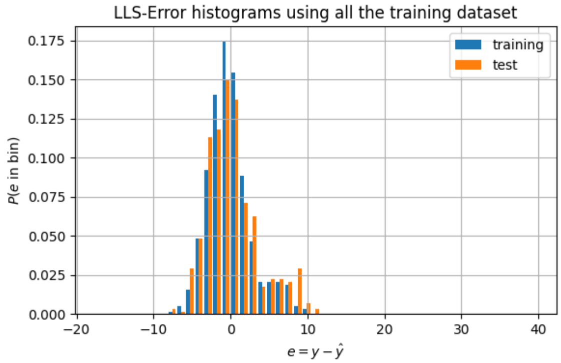
```
y_hat_te=y_hat_te_norm*sy+my
y_te=y_te_norm*sy+my
y_hat_tr=y_hat_tr_norm*sy+my
y_tr=y_tr_norm*sy+my
```

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.

```
E_tr=(y_tr-y_hat_tr)# training
E_te=(y_te-y_hat_te)# test
e=[E_tr,E_te]
plt.figure(figsize=(6,4))
plt.hist(e,bins=50,density=True, histtype='bar',
label=['training','test'])
plt.xlabel(r'$e=y-\^y$')
plt.ylabel(r'$P(e$ in bin$)$')
plt.legend()
plt.grid()
plt.title('LLS-Error histograms')
plt.tight_layout()
plt.savefig('./LLS-hist.png')
plt.show()
```

Performance figures [2]



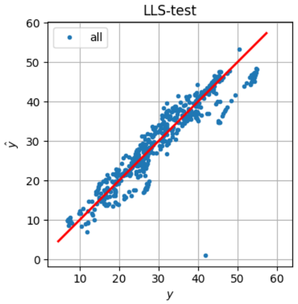
Clearly, the error does not have a Gaussian pdf, it looks like a mixture of two Gaussian pdfs, and there is not much difference in the training and test subsets (which means that there is no **overfitting**). What do you expect this figure to be like, if overfitting were present?

Performance figures [3]

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

```python
y_hat_te=(X_te_norm@w_hat)*sy+my
y_te=y_te_norm*sy+my
plt.figure(figsize=(6,4))
plt.plot(y_te,y_hat_te,'.')
v=plt.axis()
plt.plot([v[0],v[1]],[v[0],v[1]],'r',linewidth=2)
plt.xlabel(r'$y$')
plt.ylabel(r'$\^y$')
plt.grid()
plt.title('LLS-test')
plt.tight_layout()
plt.savefig('./LLS-yhat_vs_y.png')
plt.show()
```

Performance figures [4]



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:

```
E_tr_min=E_tr.min()
E_tr_max=E_tr.max()
E_tr_mu=E_tr.mean()
E_tr_sig=E_tr.std()
E_tr_MSE=np.mean(E_tr**2)
R2_tr=1-E_tr_MSE/(np.var(y_tr))
c_tr=np.mean((y_tr-y_tr.mean())*(y_hat_tr-y_hat_tr.mean()))
c_tr=c_tr/(y_tr.std()*y_hat_tr.std())
E_te_min=E_te.min()
E_te_max=E_te.max()
E_te_mu=E_te.mean()
E_te_sig=E_te.std()
E_te_MSE=np.mean(E_te**2)
R2_te=1-E_te_MSE/(np.var(y_te))
c_te=np.mean((y_te-y_te.mean())*(y_hat_te-y_hat_te.mean()))
c_te=c_te/(y_te.std()*y_hat_te.std())
```

Performance figures [6]

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

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

The printed output is

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

Performance figures [7]

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

- The error mean in the training subset is zero (as it should be); the error mean is slightly positive in the test subset 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.
- Since the error standard deviation is around 3-4 UPDRS points, this means that most of the times the regression error is around 6-8 points (twice the standard deviation, 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.

Performance figures [8]

- 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. Using motor UPDRS to predict total UPDRS makes sense only if motor UPDRS can be automatically measured using sensors. If, on the other hand, a medical doctor is needed to measure motor UPDRS, then he/she can complete the visit to provide total UPDRS too, and the regression is useless.

Table of Contents

What you have to do [1]

1. Use your ID/matricola number as random seed.
2. Drop features Jitter:DDP and Shimmer:DDA.
3. 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).
4. Then drop also Motor_UPDRS so that only voice parameters are used for regression. What happens?
5. Organize your software so that by changing the value of a variable you run your code with or without Motor_UPDRS.

Some final comments

- 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 or CUDAs, you simply 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.
- **You might be asked to run your script during the oral exam, explain what you did and why you did it and comment the results**. Therefore it is important that you appropriately store your software in organized folders.

Next lab

In the next lab we will improve the results of this lab by using only nearest neighbours to perform regression.