ICT for Health
Laboratory # 1
Regression on Parkinson data

Monica Visintin

Politecnico di Torino

2017/18

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 guts, is absorbed by the stomach; as the movements become slower and slower, levodopa stays more and more in the stomach and cannot reach the guts; at this point levodopa is directly inserted in the guts

Parkinson's disease [2]

- 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.
- 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). Adding together the scores of each single movement 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]

- CNR (Italian National Research Centre), in its Politecnico center, is working on Parkinson's using Kinect: the patient makes the movements in front of the Kinect camera, the software analyses the video and gives the score. Some of the movements that give rise to the total UPDRS have been considered, many are still to be included.
- In the literature, articles have been published in which the authors claim that it is possible 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. The claim is not so correct, since Parkinson's disease not always affects voice. We want to try and reproduce the experiments using public data.

Table of Contents

Prepare and view the data [1]

- Download from `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/`
- File `parkinsons_updrs.data` stores F columns and N_T rows; the columns are separated by commas.
- For each patient (identified by an integer in the first column), the interesting features are stored in columns from 5 to 22 (in Python columns from 4 to 21)
- Column 4 (3 for Python) stores the time (in days) at which the measurements were taken during the study, which lasted 6 months
- In each measurement day, the physician analyzed the patient and gave him/her two grades in the Unified Parkinson's Disease Rating Scale (UPDRS):

Prepare and view the data [2]

  - the total UPDRS value (column 6 - column 5 for Python)
  - the motor UPDRS (column 5 - column 4 for Python)

  these two measurements are expensive and we would like to just estimate them using the other features.
- Columns from 7 to 22 (from 6 to 21 for Python) store these other features. Each feature was actually measured 5-6 times during each measurement day, and the files stores each of these 5-6 values (not the average).
- For each patient, time goes from approximately 0 to approximately 180 days 5-6 times in subsequent blocks of data; since total and motor UPDRS values were measured just once in the day, the UPDRS values are just copied in each of the 5-6 blocks, whereas each block stores a different value for each of the other features.

Prepare the data

- **Prepare a new matrix** in which column 4 (test times, column 3 for Python) goes just once from 0 to 180 for each patient, and the values stored in columns from 7 to 22 (from 6 to 21 in Python) are the means of the values stored in the 5-6 blocks of the original file. You should get a new matrix `data` with 990 rows (instead of the 5876 present in `parkinsons_updrs.data`).

Perform regression [1]

- Consider only the data related to the first 36 patients (leave patients from 37 to 42 for the testing phase of the learning machine), and store these data into a new matrix `data_train`. Define the matrix `data_test` with the data of patients from 37 to 42.
- It is convenient to normalize the data, so that each feature has mean zero and variance 1; for each column of the matrix (for each feature) evaluate the mean and subtract it from the column, then evaluate the variance and divide the zero-mean column by the square root of the measured variance. Check that the obtained matrix has zero mean and unit variance for each feature/column
- Generate `data_test_norm` by normalizing `data_test` using the measured mean and variance of the training data

Perform regression [2]

- Define `F0` as the feature that we want to estimate from the other features; then generate the column vector `y_train=data_train_norm[:,F0]` and the matrix `X_train` from `data_train_norm` by removing column `F0`
  similarly define `y_test=data_test_norm[:,F0];` and `X_test` from matrix `data_test_norm` by removing column `F0`
- start by regressing jitter% (column 6 for Python) from the other features; once this reression works, try and regress total UPDRS.
- Perform regression using
  1. MSE,
  2. the gradient algorithm,
  3. the steepest descent algorithm
  4. the ridge regression (optimize λ)

  In the last two cases, start with a random vector $\hat{\mathbf{w}}(0)$, but generate the same random vector each time you run your script (find out how to do this)

Perform regression [3]

- Generate at least the following plots:
  1. `yhat_train` versus `y_train`
  2. `yhat_test` versus `y_test`
  3. the histograms of `y_train-yhat_train` (50 bins)
  4. the histograms of `y_test-yhat_test` (50 bins)
  5. the values of $\mathbf{w}$
- Write the report as explained in the next slides. Note that you'll be asked to give in only the final report at least 3 days before the exam, not the intermediate ones. Therefore you can take your time for the writing of the report, but we suggest to start writing it when the software is still fresh in your mind (otherwise you waste your time: first you think of the solution and then, after a couple of months, you have to re-think why you used that solution)

Table of Contents

The final report [1]

- The final report shall collect the reports of each laboratory
- The same data set will be used in more than one laboratory and the final report shall include comparisons and comments on the results obtained with the various methods/techniques/algorithms
- Minimum requirements of the report:
  1. The report shall have a first page with: the subject (**Final report on the ICT for Health laboratories**), the **author**, the academic year. If you want, you can include other details like PoliTo logo, ICT for Smart Society logo, etc.
  2. The report shall be a **pdf file** with name `report_authorsurname.pdf`, where `authorsurname` is your surname.
  3. **Spell checking** shall be used in order to remove most of the typos. Careful reading is suggested, to remove the remaining typos. If you are not sure of a word, search for it on the web/dictionary...
  4. Never include **figures** that are not referred to in the text

The final report [2]

5. Never write sentences like "In the figure below" or similar. Give **numbers to figures** and refer to them in the text through the figure number. The figure number can be part of the caption (if you use Latex or Word) or part of the title of the plot (the figure number is included in the Python code that generates the figure itself) if you use Jupyter

- The final report can be organized in several ways but we suggest one chapter for each data set, and a section for each laboratory that used that data set.
- For each data set, you must include a short description of the set itself (how many patients, how many features, if data are missing or not, if the features are numeric or categorical, etc) and the purpose of the analysis (regression, clustering or classification, and what is the logic behind the analysis, why we are doing the analysis).

The final report [3]

- For each laboratory, you must include a short description of the method/algorithm you used, the results and the comments (10 lines are typical for the comments) with the comparison with other methods, if applicable.
- The Python **scripts** shall be sent together with the report:
  - you can build a main folder that includes a subfolder for each data set, zip and send the main folder
  - you can include the scripts in the report (this is the natural choice if you use Jupyter)
- The Python scripts shall include **comments** in English (not in Spanish or Italian), to help understand the flow of instructions
- The plots shall have the **xlabels** and **ylabels**, the **grids**, the **title**. If you want to compare two plots, use the **same ranges** for the x and y axes

Table of Contents

Python-Pandas [1]

- A useful Python library that can be used to analyze data is `Pandas`: download it
- Start a new script in which you import pandas (as pd), matplotlib.pyplot (as plt), NumPy (as np)
- Read the Parkinson's data file using Pandas function `pd.read_csv(''filename'')` . The instruction is:
  `x=pd.read_csv("parkinsons_updrs.csv")`
  It is not necessary to use objects in the current laboratory, and this instruction can be written directly below the import instructions.
- 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`). For example:
  1. `x.info()`
     gives you information about the data

Python-Pandas [2]

2. `x.describe()`
   gives you the statistical description of the data set (min, max, mean, etc of each feature)
3. `x.plot()`
   plots the data
4. `x.plot.hist(bins=50)`
   plots the histograms of all the columns of the DataFrame, using 50 bins
5. `x.plot.scatter()`
   plots the scatter plot, i.e. the data of column `i` versus the data of column `k`, for any couple `i,k`. This is a very powerful tool to visually understand if two columns/features are correlated.
6. `x.values()`
   gives you the NumPy ndarray with the data

Lists and Ndarrays in NumPy [1]

- In NumPy, a list is an array of numbers or characters or other more complex objects; you cannot perform math operations on lists; the objects in the list can be arrays, not necessarily with the same length
- In NumPy, an Ndarray (**N-d**imensional **array**) is an `N`-dimensional array (a linear algebra vector if `N`=1, a linear algebra matrix if `N`=2, stacked matrices if `N`=3, etc.), and you can perform mathematical operations on these structures. A 2-dimensional Ndarray has `Nr` rows and `Nc` columns, and all the columns have `Nr` elements, all the rows have `Nc` elements (lists do not require the same lengths, on the contrary).
- If `x` is a NumPy Ndarray, `x.ndim` is its dimension (1 for linear algebra vectors, 2 for linear algebra matrices, etc), `x.shape` is `(Nr,Nc)` for a 2-dimensional Ndarray, `x.size` is the number of elements of the Ndarray (for dimension 2, it is equal to `Nc*Nr`), `x.dtype` is the type of the elements (integers, float, etc). Moreover `type(x)` gives you the answer `'NumPy.ndarray'`.

Lists and Ndarrays in NumPy [2]

- If, in Python, you write
  `a=[1, 2, 3, 4]`
  then `a` is a list. If you want an Ndarray, you must write
  `x=np.array([1, 2, 3, 4])`
  (essentially, you ask Python to change the list [1,2,3,4] into a NumPy Ndarray, that is why you need the square brackets within the round brackets).
- To create an Ndarray with all zeros or all ones or empty (you don't care what the content is), you can write:

```
z=np.zeros((Nr,Nc),dtype=np.int16)# 16 bit integers,
for example
o=np.ones((Nr,Nc),dtype=np.float64)# double precision
float, for example
e=np.empty((Nr,Nc))
```

Lists and Ndarrays in NumPy [3]

- To create an Ndarray with linearly increasing values, you can use:
  `c1=np.arange(i1,i2,istep)# from i1 to i2 with step istep`
  `c2=np.linspace(i1,i2,Nitem)# Nitem values from i1 to i2, included`
- Ndarray multiplications. DIFFERENCE WITH RESPECT TO MATLAB!!! If `A` and `B` are two 2-dimensional Ndarrays, then
  1. `A*B` is the element by element multiplication (in Matlab this is A.*B); `A` and `B` must have exactly the same shape
  2. `A.dot(B)` or `np.dot(A,B)` is the product of the two matrices (in Matlab A*B); if `A` has `Nc` columns, `B` must have `Nc` rows.
- Matrix inversion: `np.linalg.inv(A)` gives you the inverse of a square matrix, `np.linalg.pinv(A)` gives you the pseudo-inverse of a rectangular matrix (i.e. $(A^T A)^{-1} A^T$)

Lists and Ndarrays in NumPy [4]

- Other operations: if `a` is an Ndarray with shape (6,2), then
  `a.T` is the transpose of `a`
  `a.reshape(3,4)` is a new Ndarray with shape (3,4) (by rows, but check!)
  `a.ravel()` is a new Ndarray with shape (1,12) (by rows, but check!)
  `a.resize(3,4)` changes `a` (it does not generate a new Ndarray in new memory)
- If `A` is a NumPy Ndarray with 10 rows and 8 columns and you want to remove the fourth column, you can write
  `Ad=np.delete(A,3,1)`
  If you want to remove the fourth row, you can write
  `Ad=np.delete(A,3,0)`

Lists and Ndarrays in NumPy [5]

- Indexing (i.e. reading/writing some elements of an Ndarray): if `a` is a 1-dimensional Ndarray, then
  `a[0]` is the first element of `a` (in Matlab it is a(1))
  `a[1]` is the second element of `a` (in Matlab it is a(2))
  `a[-1]` is the last element of `a` (in Matlab it is a(end))
  If b is a 2-dimensional Ndarray, then
  `b[0,0]` is the element in the first row, first column (in Matlab it is b(1,1))
- Slicing (i.e. selecting portions of Ndarrays): if `a` is a 1-dimensional Ndarray, then
  `a[2:5]` is the smaller Ndarray with 3 elements, namely `a[2]`, `a[3]`, `a[4]` (in Matlab you would write a(2:4))
  If b is a 2-dimensional Ndarray, then
  `b[:,0]` is the first column (in Matlab it is b(:,1))
  `b[:,1]` is the 2-nd column (in Matlab it is b(:,2))
  `b[0,:]` is the first row (in Matlab it is b(1,:))

Lists and Ndarrays in NumPy [6]

`b[-1,:]` is the last row (in Matlab it is b(end,:))

- If in Python `A` is an Ndarray (assume it has dimension 1, for simplicity) and you write `B=A`, then the address of the first memory cell of the Ndarray `A` is copied in `B` (Python does not copy matrix `A` into a new portion of memory to create the new matrix `B`); this means that, if you write:
  `A=B`
  `B[1]=3`
  then `A[1]` is also equal to 3. This property may be useful when you want to change a portion of an Ndarray; assume for example that `A` is a (3,5) Ndarray and you want the second column to have all ones, then you can write
  `s=A[:,1]`
  `s[:]=1`
  or you can write
  `A[:,1]=1`

Lists and Ndarrays in NumPy [7]

When the Ndarray has many dimensions, the possibility of writing simply `s` instead of `A[:,1]` might be useful.

- If you want to copy Ndarray A into a new Ndarray B (i.e. a new portion of memory), then you must write
  `B=A.copy()`
  or `B=1*A`

Series and DataFrames in Pandas [1]

- In Pandas, a Series is a one-dimensional object, a DataFrame is a two-dimensional object, made of Series. We will typically use DataFrames.
- One of the main differences between DataFrames and Ndarrays is that in DataFrames, you can access columns of data according to their name, not only their index. If in DataFrame `x` you have a column labeled `'age'`, then `z=x.loc(:,'age')` generates the series `z` with this column, and you don't have to bother where column `'age'` is in the data set.
  If `x` is a DataFrame with columns `'c1'`, `'c2'`, , `'c10'`, and you want to split it into two smaller DataFrames, you can use:
  ```
  x1=x.loc[:,'c1':'c5']
  x1=x.loc[:,'c6':'c10']
  ```