ICT for Health
Laboratory # 1
Regression on Parkinson data

Monica Visintin

Politecnico di Torino

2018/19

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 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 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.

- For each patient, time goes from approximately 0 to approximately 180 days 5-6 times in subsequent blocks of data; for a given date you have one measurement of the motor UPDRS, one measurement of the total UPDRS and some measuremnts of the voice parameters in 4-5 rows (check it on your own).
- **Randomly permute (shuffle) the rows of the matrix** and call the matrix `data`.

Perform regression [1]

- Use the first 50% of the rows of `data` as **training points** (define the submatrix as `data_train`), the subsequent 25% as **validation points** (define the submatrix as `data_val`) and the remaining 25% of the rows as **testing points** (define the submatrix as `data_test`.
- 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. Prove that, if you normalize the data, it is no more necessary to add a **column of all ones** to `data`. Write the proof in the report.
- Generate `data_test_norm` by normalizing `data_test` using the measured mean and variance of the training data (previous point); similarly generate `data_val_norm`

Perform regression [2]

- Define `F0` as the feature that we want to estimate from the other features (**regressand**); then generate the column vector `y_train=data_train_norm[:,F0]` and the matrix `X_train` from `data_train_norm` by removing column `F0`, `X_test` from matrix `data_test_norm` by removing column `F0`, `X_val` from matrix `data_val_norm` by removing column `F0`; moreover define `y_test=data_test_norm[:,F0]; y_val=data_val_norm[:,F0];`
- start by regressing **shimmer** from the other features; once this reression works, try and regress **total UPDRS**. **In the report only include the results for total UPDRS**.
- Perform regression using
  1. LLS pseudoinverse
  2. the conjugate gradient
  3. the gradient algorithm,
  4. The stochastic gradient
  5. the steepest descent algorithm
  6. the ridge regression (optimize λ)

Perform regression [3]

In the last 4 cases, start with a random vector $\hat{\mathbf{w}}(0)$, but generate the same random vector each time you run your script (**set the seed** in Python)

Perform regression [4]

- 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 $\mathbf{w}$

  Note that you have to **show the un-normalized data**. Add all the plots you think adequate to describe your work.
- Write in the report the measured **mean square errors** for the training dataset, the validation dataset, the testing dataset.
- Write the **report** as explained in the slides at the end of this file.
- **Deadline for the report: November 7th 2018, Wednesday, at 11.59 PM.**

Table of Contents

Slicing and indexing in Numpy and Pandas

In the lab, you need to select portions of matrices and play with them.
You can use Pandas or you can use Numpy to perform **indexing** (i.e. find the index of soimething you are interested in) and **slicing** (i.e. selecting the portion of matrices/vectors/arrays you are interested in, for further processing).

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")`
- 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; use for example `Data=x.values()` to get the Ndarray `Data` if you want to mainly use Numpy in the lab.

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 the **shape** `(Nr,Nc)` for a 2-dimensional Ndarray, `x.size` is the **size**, i.e. 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` `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)
- sometimes you might get an Ndarray `a` with shape `(N,)`, that is **not a column nor a row.** You can set it to a column vector by writing `a.shape=(N,1)`, to a row vector by writing `a.shape=(1,N)`.

Lists and Ndarrays in NumPy [4]

- 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)`
- **Indexing** (i.e. find the position of an element in 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))

Lists and Ndarrays in NumPy [5]

- **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,:))
  `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
  ```

Lists and Ndarrays in NumPy [6]

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
```

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']
  ```
- However you can also use iloc that uses integers (and not names): `x.iloc[1:5, 2:4]` takes the rows from 2 to 5 (4 rows, starting from row 1 as typical in linear algebra) and columns from 3 to 4 (2 columns).

Series and DataFrames in Pandas [2]

- Remember that in DataFrames the rows are identified by the `index` whereas the columns by the `columns`: to create a dataframe, you can write, for example
  `df1 = pd.DataFrame(np.random.randn(6,4),`
  `index=list(range(0,12,2)),columns=list(range(0,8,2)))`
  and you get

```
Out[67]:
            0          2          4          6
0    0.149748  -0.732339   0.687738   0.176444
2    0.403310  -0.154951   0.301624  -2.179861
4   -1.369849  -0.954208   1.462696  -1.743161
6   -0.826591  -0.345352   1.314232   0.690579
8    0.995761   2.396780   0.014871   3.357427
10  -0.317441  -1.236269   0.896171  -0.487602
```

Table of Contents

The report [1]

- 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. Once you are ready, generate a **pdf** file with your report and a folder with all the Python code. Zip together the folder and the report pdf file and generate a file `yoursurname_yourmatricola_report_1.zip` that you **upload in folder "Elaborati"** of the class webpage (you access it from your PoliTo webpage). The file name could be, for example, `visintin_s34527_report_1.zip`. IF THE FILENAME IS NOT AS DESCRIBED THE GRADE IS ZERO.
  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 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. Learn how to do it in Word/Latex.
6. 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
7. The Python scripts shall include **comments** in English (not in Spanish or Italian), to help understand the flow of instructions

- The report must be organized in **sections**: one section for the introduction, one section for each of the methods/algorithms, one section for the conclusions and comparisons.
  - Introduction shall give the description of the problem (Parkinson's disease) the available data (number of patients, features, etc) and the goal (what we want to regress and why)
  - Each section describing the method/algorithm shall include a brief description of the method itself (with the formulas if they are necessary, but for sure with the description of the parameters and the values that you used) and the results; typically 10 lines to comment the results are enough

The report [3]

- the section with the conclusions shall include the comparison among the various methods and your personal comments.
- You cannot use symbols (like γ), without saying what they mean (the learning coefficient). A student from another university would not understand what γ is (maybe in this other university they use α). Never use **acronyms** without specifying the meaning.
- It is not necessary that you give the details of the gradient algorithm, for example, the algorithm is available on the Web, you assume the reader knows it, or he/she can learn it from the web.
- The expected **number of pages** is maximum 15, A4 page, using font 12pt. Figures can be compressed by showing for example more curves in the same figure, provided that the curves can be easily read and understood.
- A missing report corresponds to **grade** 0, an incomplete report to a grade between 0 and 4 (it depends on the portion of missing material), a complete report to a grade from 0 to 5 (it might be completely wrong), a report with many errors in the use of English a grade from 0 to 4.