TP1_LinearRegression

September 24, 2021

1 TP1 : Linear regression

The purpose of this work is to implement least square linear regression to medical data. The problem is based on an example described in the book by Hastie & Tibshirani (2009) pp. 3-4 & 49-63. Data come from a study published by Stamey et al. (1989). This study aims at the prediction of the level of prostate specific antigen, denoted by `lpsa` below, from the results of clinical exams. These exams are carried out before a possible prostatectomy.

The measurements are log cancer volume `lcavol`, log prostate weight `lweight`, age of the patient `age`, log of benign prostatic hyperplasia amount `lbph`, seminal vesicle invasion `svi`, log of capsular penetration `lcp`, Gleason score `gleason`, and percent of Gleason scores 4 or 5 `pgg45`. The variables `svi` and `gleason` are categorical, others are quantitative. There are `p=8` entries. The work is decomposed in the following tasks:

- read and format the data : extraction of the training and test sets,
- apply least square regression method to predict `lpsa` from the entries,
- study the estimated error on the test set (validation),
- identify the most significant entries by using a rejection test,
- apply regularized least square regression method (ridge regression),
- search for an optimal regularization parameter thanks to cross-validation.

[ ]:
```
import csv
# import os
from pylab import *
import numpy as np
from numpy import linalg as la

import pandas as pd
import seaborn as sns
```

1.1 Read & Normalize data

Data are stored in ASCII format:

- the first column enumerates the data from 1 à 97 (97 male subjects).
- columns 2 to 9 contain the entries themselves.
- column 10 contains target values.
- column 11 contains label 1 for the training set, and 2 for the test set.

[ ]:
```
#%% To read data from spaced separated float numbers
# x, y = np.loadtxt(c, delimiter=',', usecols=(0, 2), unpack=True)

data_init = np.loadtxt('prostate_data_sansheader.txt')

data = data_init[:,1:]   # we get rid of the indices (1 to 97)

#%% Extraction of training/test sets
Itrain = np.nonzero(data[:,-1]==1)
data_train=data[Itrain]   # original data

Itest = np.nonzero(data[:,-1]==0)
data_test = data[Itest]   # original data
```

Normalization of the data *with respect to the mean and standard deviation of the training set.*

[ ]:
```
M_train = data_train
M_test = data_test
moy = np.zeros((8,))
sigma = np.zeros((8,))

# With a FOR loop :
for k in range(8): # 8 columns of entries
    moy[k]=np.mean(data_train[:,k])
    sigma[k] = np.std(data_train[:,k], ddof=0)
    M_train[:,k] = (data_train[:,k]-moy[k])/sigma[k] # normalized: centered,␣
 ↪variance 1
    M_test[:,k] = (data_test[:,k]-moy[k])/sigma[k]   # same normalization for␣
 ↪test set

print(sigma)
```

[ ]:
```
# Alternative WITHOUT FOR
normalize = lambda vec: (vec-np.mean(vec))/np.std(vec)    # inline function
M_train = np.array( [ normalize(vec) for vec in data_train[:,0:8].T ] ).T  #␣
 ↪iterate on vec direct / ARRAY not LIST
moy = np.array( [ np.mean(vec) for vec in data_train[:,0:8].T ] )
sigma = np.array( [ np.std(vec, ddof=0) for vec in data_train[:,0:8].T ] )

M_test = np.array([ (data_test[:,k]-moy[k])/sigma[k] for k in range(M_train.
 ↪shape[1]) ] ).T
```

2 Part 1 : simple least square regression

2.1 Preliminary questions

- Compute the autocovariance matrix from the training set.

- Observe carefully & Comment. What kind of information can you get ?

[ ]:
```
# Preliminary questions
```

Quelques rappels à propos de la notion de covariance : https://fr.wikipedia.org/wiki/Covariance

2.2 Exercise 1 : least square regression

- Build the matrix of features `X_train` for the training set, the first column is made of ones.
- Estimate the regression vector `beta_hat` (estimates= `X*beta_hat`) *Indication: you may either use the function* ***inv*** *or another more efficient way to compute* $A^{-1}B$ *(think of* `A\B`*).*
- What is the value of the first coefficient `beta_hat[0]` ? What does it correspond to ?
- Estimate the prediction error (quadratic error) from the test set.

Indication: be careful of using ***X_test*** *defined above, normalized w.r.t. the training data set. You can estimate this error by using:*

[ ]:
```
t_test = data_test[:,8]   # target column
N_test = data_test.shape[0]
X_test = np.concatenate((np.ones((N_test,1)), M_test[:,0:8]), axis=1)
# don't forget the 1st column of ones and normalization !
```

[ ]:
```
# Exercise 1
```

2.3 Rejection test, computation of Z-scores

Now we turn to the selection of the most significant entries so that our predictor be more robust. The essential idea is that our estimates will be more robust if only the most significant entries are taken into account. As a consequence, note that we will *reduce the dimension* of the problem from |p=8| to some smaller dimension. The present approach uses a statistical test to decide whether the regression coefficient corresponding to some entry is significantly non-zero. Then we can decide either to put non significant coefficients to zero, or to select the significant entries only and estimate the new reduced regression vector.

Let's assume that target values are noisy due to some white Gaussian noise with variance σ^2 (see Hastie & Tibshirani p. 47). One can show that the estimated regression vector |beta_hat| is also Gaussian with variance

$$var(\widehat{\beta}) = (X^T X)^{-1}\sigma^2.$$

One can also show that the estimator of the variance (from the training set)

$$\widehat{\sigma^2} = \frac{1}{(N-p-1)}\sum(t_n - \widehat{t}_n)^2$$

obeys a Chi-2 distribution. As a consequence a Chi-square statistical test can be used to determine whether some coefficient β_j is significantly non-zero. To this aim, one defines the variables z_j named Z-scores which in turn obey a Fisher law, also called t-distribution, which are often used in statistics:

$$z_j = \frac{\beta_j}{\widehat{\sigma}\sqrt{v_j}}$$

where v_j is the j-th diagonal element of the matrix $(X^T X)^{-1}$. For sake of simplicity, we will consider that the null hypothesis of β_j is rejected with probability 95% if the Z-score is greater than 2.

2.4 Exercise 2

1. Compute the Z-scores and select the most significant entries.
2. Estimate the prediction error over the test set if only these significant entries are taken into account for regression by putting other regression coefficients to zero.
3. Estimate the new regression vector when only the significant features are taken into account.
4. Compare to previous results (Exercise 1).

Indication 1 : to sort a vector Z in descending order `val = np.sort(np.abs(Z))[-1:0:-1]`

Indication 2 : to extract the diagonal of a matrix, `vXX = np.diag(inv(X.T.dot(X),k=0)`

```
[ ]: # Exercise 2
```

3 Part 2: Regularized least squares

This part deals with regularized least square regression. We denote by `beta_hat_reg` the resulting coefficients. This approach is an alternative to the selection based on statistical tests above. The idea is now to penalize large values of regression coefficients, *except for the bias.*

We use the result:

$$\hat{\beta} = (\lambda I_p + X_c^T X_c)^{-1} X_c^T t_c$$

where X_c contains the normalized entries of the training data set with no column of ones (the bias should no be penalized and is processed). The targets `t_c` are therefore also centered, `t_c=t-mean(t)`.

First, we estimate the bias t_0 to center the targets which yields the coefficient β_0, that is `beta_hat_reg[0]` in Python.

Remark : the bias is estimated as the empirical average of targets. For tests, entries should be normalized with respect to the means and variances of the training data set (see exercise 3.5 p. 95 in Hastie & Tibshirani). Then work on the vector of entries with no column of ones.

3.1 Exercise 3

1. Use *ridge regression* for penalty `lambda = 25` to estimate the regression vector.
2. Estimate the prediction error from the test set.
3. Compare the results (coefficients β, error…) to previous ones.
4. You may also compare these results to the result of best subset selection below:

`beta_best = [2.477 0.74 0.316 0 0 0 0 0 0].`

Indication : a simple way to obtain predictions for the test data set is the code below:

[ ]:
```
t = data_train[:,8]   # column of targets
t0 = np.mean(t)

N_test = data_test.shape[0]
X_test = np.hstack((np.ones((N_test,1)), M_test[:,0:8]))
# Here the 1st column of X_test is a column of ones.
t_hat_reg = X_test.dot(beta_hat_reg)
```

[ ]:
```
# Exercise 3
```

4 Part 3: Cross-Validation

4.1 How to choose lambda from the training data set only ?

The idea is to decompose the training set in 2 subsets: one subset for linear regression (say 9/10 of the data), the other to estimate the prediction error (say 1/10 of the data).

We can repeat this operation 10 times over the 10 possible couples of subsets to estimate the average prediction error. We will choose the value of `lambda` which minimizes this error. The algorithm goes as follows:

For the 10 cross-validation cases

```
Extraction of test & training subsets `testset` & `trainset`

For lambda in 0:40
    Estimate `beta_hat` from normalized `trainset` (mean=0, var=1)
    Estimate the error from  `testset`
EndFor lambda
```

EndFor 10 cases

Compute the average error for each lambda

Choose `lambda` which minimizes the error

4.2 Exercise 4

- Use 6-fold cross-validation in the present study to optimize the choice of `lambda`. Try values of `lambda` ranging from 0 to 40 for instance (0:40).
- Plot the estimated error as a function of `lambda`.
- Propose a well chosen value of `lambda` and give the estimated corresponding error on the test set.
- Comment on your results.

Indication 1 : think of shuffling the dataset first.

Indication 2 : you can build 6 training and test subsets by using the code below

[ ]:
```
lmax = 40
lambda_pos = arange(0,lmax+1)

N_test = 10
m=np.zeros(8)
s = np.zeros(8)
X_traink = np.zeros((X_train.shape[0]-N_test,8))
X_testk = np.zeros((N_test,8))
erreur = np.zeros((6,lmax+1))
erreur_rel = np.zeros((6,lmax+1))
```

[ ]:
```
# Exercise 4
for p in range(6):   # loop on test subsets
    # extraction of testset
    testset  = data_train[arange(p*N_test,(p+1)*N_test),0:9]
    # extraction of trainset
    trainset =␣
 ↪data_train[hstack((arange(p*N_test),arange((p+1)*N_test,data_train.
 ↪shape[0]))),0:9]
    # normalization of entries,
    # ...
```

[ ]:
```
# Exercise 4 ---------------
# ...
# averaged error on the 6 training/test sets ?
# averaged error on the 6 training/test sets ?
# standard variation of this error estimate ?

# print(erreur_lambda, std_erreur_lambda, erreur_rel_lambda)
```

[ ]:
```
# Exercise 4 FIGURE ---------------
# ...
```

[ ]:
```
# Exercise 4 (continued)
```