Intelligent Systems 1 - Summer Semester 2015

Exercise Sheet 4, Regularized Linear Regression

submission date: 2015-05-21 (Beginning of next lecture)

2015-05-07

1 LASSO and ℓ_0 estimator for orthonormal data (30 points)

Let $X \in \mathbb{R}^{n \times p}$ be your training input set, and $Y \in \mathbb{R}^n$ your training output. In a linear model we predict $\hat{y}(x) = x^T \hat{\beta}$ for some input $x \in \mathbb{R}^p$ and some constant $\hat{\beta} \in \mathbb{R}^p$. The so-called ℓ_0-penalized estimator $\beta^0(\lambda)$ tries to find a good β with reslatively few non-zero entries. It is defined as:

$$\beta^0(\lambda) := \arg\min_\beta \frac{1}{n} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_0 , \tag{1}$$

where $\|\beta\|_0 := \sharp\{j : \beta_j \neq 0\}$ and $\lambda > 0$. Alternatively, the LASSO estimator $\beta^{\text{LASSO}}(\lambda)$ is defined as:

$$\beta^{\text{LASSO}}(\lambda) := \arg\min_\beta \frac{1}{n} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_1 , \tag{2}$$

where $\|\beta\|_2^2 := \sum_{i=1}^p \beta_i^2$, $\|\beta\|_1 := \sum_{i=1}^p |\beta_i|$ and $\lambda > 0$.

There are in general no analytic formulae for $\beta^0(\lambda)$ and $\beta^{\text{LASSO}}(\lambda)$. In practice, they are computed by numerical optimization. However, we can compute an analytic expression in the special case of *orthogonal design*, where

$$p = n \quad \text{and} \quad \frac{1}{n} X^T X = I_{p \times p}. \tag{3}$$

The goal of this exercise is to compute these formulae.

1.1 ℓ_0-regularized estimator in the orthonormal design

Let $g_{\text{hard},\lambda}$ be the function:

$$g_{\text{hard},\lambda} : \mathbb{R} \to \mathbb{R}, \quad z \to z\, 1_{\{|z|>\lambda\}}, \tag{4}$$

where $1_{\{|z|>\lambda\}} = 1$ if $|z| > \lambda$ and 0 otherwise. Suppose we are in the orthonormal design.

1. Plot $g_{\text{hard},\lambda}$ for $\lambda = 1$.
2. Write down the analytical formulae for the least square estimator $\hat{\beta}^{\text{LS}}$. (Not needed for the subsequent calculations)
3. Prove that $\beta_j^0(\lambda) = g_{\text{hard},\sqrt{\lambda}}(z_j)$ where, again, $z_j := \frac{1}{n}(X^T Y)_j$.

1.2 LASSO estimator in the orthonormal design

Let $g_{\text{soft},\lambda}$ be the function:

$$g_{\text{soft},\lambda} : \mathbb{R} \to \mathbb{R}, \quad z \to \text{sign}(z)\,(|z| - \lambda)_+. \tag{5}$$

Suppose we are in the orthonormal design.

1. Plot $g_{\text{soft},\lambda}$ for $\lambda = 1$.
2. Prove that $\beta_j^{\text{LASSO}}(\lambda) = g_{\text{soft},\frac{\lambda}{2}}(z_j)$, where $z_j := \frac{1}{n}(X^T Y)_j$.

2 Application of LASSO on a gene data set (70 points)

(This exercise is independent of the first one.)

The goal of this exercise is to apply both ridge regression and LASSO on real data. The data can be found on the homepage of the course, files *xtrain_xxxx.csv* and *xtest_xxxx.csv* , and comes from the paper [1]. It consists of the logarithm of the gene expression levels of yeast cells.

More specifically, we consider two datasets corresponding to two different targets: say gene 4710 for the first and gene 3290 for the second. In both cases, we would like to predict the gene expression levels of the target gene, given the gene expression levels of the other 6170 genes. For each dataset, we are given a training set $X_{\text{train}} \in \mathbb{R}^{140 \times 6170}$, $y_{\text{train}} \in \mathbb{R}^{140}$ consisting of 140 yeast cells, and a test set $X_{\text{test}} \in \mathbb{R}^{20 \times 6170}$ with 20 cells. To evaluate the performance of our model, we also provide the target values $y_{\text{test}} \in \mathbb{R}^{20}$ of the test set.

You are asked, to hand in:

1. A printout of the code of the function *printOutput*, that you will have appropriately modified (see section *The printOutput function*).
2. Answer questions a. and b. of the section *Linear regression with no regularizer*.
3. A printout of all the lines marked *### CHANGE THIS LINE ###*. On the printout, all regressions should be done using a regularizing parameter λ that you will have optimized using 10-fold cross-validation on the training set. However, you do not need to hand in the printout of the lines corresponding to these cross-validations.
4. A printout of the output of all lines marked *### PROVIDE THE OUTPUT ###*.
5. The same printouts as for questions 3. and 4., but with target gene 3290 instead of 4710 (i.e. with the second dataset).

Hint: For all the regression tasks, we recommend to use python functions of the type *linear_model.xxx(xxx)*, of the pacakge *sklearn*. The code asked to hand in should look pretty similar to the code of the section *Linear regression with no regularizer*. Cross-validation can be easily done using ipython functions of the type *linear_model.xxxCV(xxx)*. Do not hesitate to consult the documentation of these functions. Finally, note that, what we call λ corresponds to the parameter *alpha* of the python functions.

2.1 Loading the required packages.

```
>>> %matplotlib inline
... from numpy import *
... from scipy import *
... from matplotlib import pyplot as plt
... from sklearn import linear_model
... from pandas import * # for easy import of data
```

2.2 Target: gene 4710

2.2.1 Loading the data

```
>>> ### CHANGE THE PATHS ###
... Xtrain = read_csv("./datasets/Xtrain_4710.csv", header = False)
... Xtest = read_csv("./datasets/Xtest_4710.csv", header = False)
... ytrain = read_csv("./datasets/ytrain_4710.csv", header = False, names = ["gene","y"])
... ytest = read_csv("./datasets/ytest_4710.csv", header = False, names = ["gene","y"])
...
... Xtrain = Xtrain.drop(Xtrain.columns[[0]],axis = 1)
... ytrain = squeeze(ytrain.drop(ytrain.columns[[0]],axis = 1))
... Xtest = Xtest.drop(Xtest.columns[[0]],axis = 1)
... ytest = squeeze(ytest.drop(ytest.columns[[0]],axis = 1))
```

2.2.2 The printOutput function

The purpose of this function is to plot the regression weights, the predicted values yhat of y against ytest, the (number of the) gene with the strongest coefficient and the so-called *coefficient of determination* R^2, defined as:

$$R^2 := 1 - \frac{\sum_{(x_i,y_i) \in \text{test set}} (\hat{y}(x_i) - y_i)^2}{\sum_i (y_i - \overline{y})^2}, \tag{6}$$

where $\overline{y}$ is the emperical mean of the y_i's in the test set.

```
>>> def printOutput(lm_, Xtest_, ytest_):   # lm_ = instance of linear_model.xxx(xxx)
...     yhat = lm_.predict()  ### CHANGE THIS LINE ###
...     plt.figure(1)
...     plt.title("Regression Weights")
...     plt.plot(lm_.coef_.T)
...     plt.figure(2)
...     plt.title('yhat vs ytest')
...     plt.plot(ytest_, yhat, 'ro')
...     plt.show()
...     # print 'R2 :', lm_.score() ### CHANGE THIS LINE ###
...     # print 'Gene with Strongest Coefficient :' , lm_.coef_.xxx ### CHANGE THIS LINE ###
...     if hasattr(lm_, 'alpha'): print 'Used Lambda :', lm.alpha # if using linear_model.xxx
...     if hasattr(lm_, 'alpha_'): print 'Lambda_ResultOfCV :', lm.alpha_ # if using linear_model.xxxCV
...     # print 'Regression Coefs :', lm_.coef_ # Can be printed out, if wanted
...     # print 'Regression Intercept :', lm_.intercept_ # Can be printed out, if wanted
```

2.2.3 Linear regression with no regularizer

a. What should happen if you tried to apply linear regression without any regularizer on this data set?

b. Does this happen with the function *linear_model.LinearReagression()* ?

```
>>> lm = linear_model.LinearRegression()
... lm.fit(Xtrain,ytrain)
...
... # printOutput(lm,Xtest,ytest)   ### PROVIDE THE OUTPUT ###

LinearRegression(copy_X=True, fit_intercept=True, normalize=False)
```

2.2.4 Ridge regression

```
>>> #lm = linear_model.Ridge(xxx)  ### CHANGE THIS LINE ###
... lm.fit(Xtrain, ytrain);
...
... #printOutput(lm, Xtest, ytest)   ### PROVIDE THE OUTPUT ###

LinearRegression(copy_X=True, fit_intercept=True, normalize=False)
```

2.2.5 LASSO regression

```
>>> # lm = linear_model.Lasso(xxx) ### CHANGE THIS LINE ###
... lm.fit(Xtrain, ytrain)
...
... # printOutput(lm, Xtest, ytest)   ### PROVIDE THE OUTPUT ###
```

```
...
... print 'Regression Coefficient of Gene 5954 : ' ### CHANGE THIS LINE ### ### PROVIDE THE OUTPUT ###

Regression Coefficient of Gene 5954 :
```

References

[1] P. Kemmeren, K. Sameith, L.A. van de Pasch, J.J. Benschop, T.L. Lenstra, T. Margaritis, E. O'Duibhir, E. Apweiler, S. van Wageningen, C.W. Ko, S. van Heesch, M.M.. Kashani, G. Ampatziadis-Michailidis, M.O.. Brok, N.A. Brabers, A.J. Miles, D. Bouwmeester, S.R. van Hooff, H. van Bakel, E. Sluiters, L.V. Bakker, B. Snel, P. Lijnzaad, D. van Leenen, M.J. Groot Koerkamp, and F.C. Holstege. Large-scale genetic perturbations reveal regulatory networks and an abundance of gene-specific repressors. *Cell*, 157:740–752, 2014.