A Octave/Matlab Tutorial for Linear Regression

Matteo Matteucci and Luigi Malagò

Pattern Analysis and Machine Intelligence, 2012
Politecnico di Milano
Document revision 1.0, May 25, 2012

1 Introduction

This tutorial will guide you though the implementation of some of the algorithms for linear regression presented in [2, Ch. 3]. The code has been tested with Octave 3.4.0, available at [1], and is going to be tested in Matlab. Feel free to email us for problems or comments about the code presented. See documetation at [2]. The *prostate cancer* dataset used in this tutorial is the same used in the book, and it can be downloaded from [3].

2 Import dataset

If you open the file `prostate.data`, you will see that the first line contains the name of the covariates and of the output variable, and a label for the last column, that identifies training set and test set. Starting from the second row, the first column contains the number of observations (instances), from column 2 to 9, there are the covariates, column 10 the output, and the last column contains a char that can be either T, for observations in the training set, and F for those in the test set. We need to import these data in Octave, in order to run different linear regression algorithms.

```
   lcavol     lweight   age lbph       svi lcpgleason pgg45 lpsa  train
1 -0.579818495 2.769459  50 -1.38629436 0 -1.38629436 6   0 -0.4307829 T
2 -0.994252273 3.319626  58 -1.38629436 0 -1.38629436 6   0 -0.1625189 T
3 -0.510825624 2.691243  74 -1.38629436 0 -1.38629436 7  20 -0.1625189 T
....
```

We load labels in the first line, and data in the remaining part of the file, using different methods.

First we will load the data using the `dlmread` function. Notice that we tell Octave to ignore the first line and the first column, by specifying the starting row and column (values are zero-based) as third and forth parameters. The second parameter specifies the separator for the columns. Next we want to remove the

[1] http://www.gnu.org/software/octave/
[2] http://www.gnu.org/software/octave/doc/interpreter/
[3] http://www-stat.stanford.edu/ tibs/ElemStatLearn/

last column (numer 10) from the matrix A, since it contains information about test and training set. The 9th column contains the output variable, that we copy in y.

```
A = dlmread("prostate.data", "\t", 1, 1)

%remove last column
A(:,10) = [];
%obtain y
y = A(:,9)
%delete y from A
A(:,9) = [];
```

In order to read the labels from the datafile, we use `fopen`, `fgelt`, and `fclose` to handle files. We read the first line, remove multiple spaces with `regexprep`, then we obtain single columns by splitting split with `strsplit` using the space as separator. Next with `cellstr` we obtain cell arrays of strings from a character array. Finally we obtain the label of the covariates and that of the output. We discard the label associated to the last column, about the observations in the training set.

```
%open file
F = fopen("prostate.data");
%read line
titles = fgetl(F);
%remove extra spaces
titles = regexprep(titles,'\s+',' ');
%split line
col = strsplit(titles,' ');
%convert to cell arrays
inputLabel = cellstr(col);
%remove 'train' label
inputLabel(11) = [];
%copy and then remove from covariates label the output label
outputLabel = col(10);
inputLabel(10) = [];
%remove first empty label
inputLabel(1) = []
%close file
fclose(F);
```

Next, we read the last column, to filter observations in the dataset, and obtain the training set. To do that, we need to read with `textread` the last column. We use this functions, since the last column contains characters. We discard the other columns.

```
%read last column of the matrix
[c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10] = textread('prostate.data',
   '%d %d %d %d %d %d %d %d %d %d %s','headerlines', 1);

%convert cell  arrays to a character array
train = char(c10);
```

If your version of Octave does not include the `textread` function, you can use the two m files provided with this tutorial `strread.m` and `textread.m`. That that version of `textread` ignores the `'headerlines'` parameter, so you have to do something like

```
%read last column of the matrix
[c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10] = textread('prostate.data',
   '%d %d %d %d %d %d %d %d %d %d %s');

%convert cell  arrays to a character array
train = char(c10);
train = train(2:end,1)
```

Now we can filter the dataset, and obtain y and X, for the training set used in the references

```
%create set of indices of the points we want to consider
idx = (train == 'T');
%filter dataset
y = y(idx)
X = A(idx,:);
```

IN THE FOLLOWING TUTORIAL AND IN THE EXERCISES AND QUESTIONS, DO NOT FILTER THE DATASET ACCORDING TO THE TRAINING SET COLUMN. USE ALL OBSERVATIONS (BOTH TRAINING SET AND DATASET) AS ONE UNIQUE DATASET. This implies that you have to comment the two previous lines If you use the training se as in the dataset, you will obtain the same results that appear in the book, however WE WILL EVALUATE THE FOLLOWING EXERCISES AND QUESTIONS WITH RESPECT TO THE COMPLETE DATASET (NO DISTINCTION BEWEEN TRAINING AND TEST SETS). You can write all your code in a m file and run it over different datasets to verify the results.

We can easily check the dimension of out vectors and matrix, and obtain the number of observations N and the number of covariates p

```
size(outputLabel)
size(inputLabel)
size(X)
size(y)
size(train)
N=size(X)(1)
p=size(X)(2)
```

3 Least Squares Estimation

In this section solve the linear regression problem with least squares estimation. First we evaluate some statistics about the covariates in the training set, such as the average and standard deviation.

```
%averages
mean(X)
%standard deviations
std(X)
%correlations between two variables
cov(X(:,1),X(:,2))/(std(X)(1)*std(X)(2))
```

Next we normalize our variables, so that they have zero mean and unit standard deviation.

```
for i = 1:p
    X(:,i) = (X(:,i) - mean(X)(i))/std(X(:,i));
endfor

%mean vector
mean(X)
%std vector
std(X)
```

Next, since variables have unit standard deviation, we can evaluate correlations by covariances.

```
%evaluation of correlations
for i = 1:p
    cov(X(:,1),X(:,i))
endfor
```

Exercise: Evaluate the correlations among all couples of variables, as in Table 3.1 of [2]

In order to include the intercept as a coefficient to be estimated by least squares, we introduce a new column of in X of all ones.

```
X = [ones(N,1) X]
```

We can now obtain the β coefficients with the usual formula, and similarly standard errors and Z-scores.

```
%evaluations of beta coefficients
beta = inv(X'*X)*X'*y

%standard errors and Z scores
sigmahat_sq = (y-X*beta)'*(y-X*beta)/(N-p-1);
XX = inv(X'*X);
for i = 1:p+1
    z = beta(i)/sqrt(sigmahat_sq*XX(i,i))
endfor
```

Exercise: Evaluate β, standard errors and Z scores as in Table 3.2 of [2]

4 Successive orthogonalization, Gram-Schmidth procedure and QR decomposition

In this section we implement the algorithm for linear regression via successive orthogonalizations, as described in Algorithm 3.1 of [2]. By orthogonalization of the inputs, we obtain a new matrix Z whose column vectors are orthogonal and they span the same space spanned by the columns vectors of X. Algorithm 3.1 describes the procedurs to obtain the $\hat{\beta}_p$ coefficient of x_p, however such procedure can be easily generalized to obtain the complete β vector. Indeed, it is easy to see that the z_j are obtained as a linear combination of the x_j, which can be written using the matrix notation. More in details, the γ coefficients obtained at step 2 of the algorithm form the upper triangular matrix Γ, such that $X = Z\Gamma$, see formula 3.30 of [2]. From step 2 of the algorithm, by expressing x_j as a function of the $z_1, \dots, z_j$, we have that the $gamma_{j\ j}$ are all equals to 1. By decomposing X as $ZD^{-1}D\Gamma$, where D is diagonal and $D_{jj} = ||z_j||$ as in Equation 3.31, we obtain a QR decomposition of X, with $Q = ZD^{-1}$ and $R = D\Gamma$. Then, by plugging such decomposition in the formula for $\hat{\beta}$, and observing that $Q^TQ = I$, we obtain

$$\begin{aligned}\hat{\beta} &= (X^{-1}X)X^Ty \\ &((QR)^TQR)^{-1}(QR)^Ty \\ &(R^TQ^TQR)^{-1}R^TQ^Ty \\ &R^{-1}(R^T)^{-1}R^TQ^Ty \\ &R^{-1}Q^Ty.\end{aligned}$$

This is just another approach to solve LSE, using the QR decomposition, also known as Gram-Schmidth procedure. The results obtained are usually more numerically stable, due to reduced conditions numbers, that's why this method sometimes works better than regular formula for LSE.

```
%initialize Z
```

```
Z = [ones(N,1)];
%initialize G
G = [];
%fix first column of G (see the next question, after the code)
gamma = [1];
for i = 1:p
    gamma = [gamma 0];
endfor;
G = [gamma']

for j = 2:p+1
    gamma = [];
    %regress x_j on z_1, ..., z_j-1 (rows and columns are one-based)
    for l = 1:j-1
        gamma = [gamma; (Z(:,l)'*X(:,j))/(Z(:,l)'*Z(:,l))];
    endfor
    %update Z by adding z_j obtained with by linear combination of
    %x_j and z_1, ..., z_{j-1}
    Z  = [Z X(:,j)-Z*gamma];
    %add 1 on the diagonal of G (see previous comment about step 2 of
    %the Algorithm
    gamma = [gamma; 1];
    %add zeros in the remaining components of the vector
    for i = j+1:p+1
        gamma = [gamma; 0];
    endfor
    gamma;
    %update G
    G = [G gamma];
endfor

G
Z
%estimate the beta coefficient associated to the last variable x_{p+1} used
%in the regression (p+1, since x_1 = [1,...,1] )
beta_p = (Z(:,p+1)'*y)/(Z(:,p+1)'*Z(:,p+1))

%create D matrix
D = zeros(p+1,p+1);
for j=1:p+1
    D(j,j) = norm(Z(:,j));
endfor

%check sizes
size(D);
size(G);
size(D*G);

%evaluate complete beta vector by QR decomposition
newBeta = inv(D*G)*(Z*inv(D))'*y
```

Question: Why does the first column of Γ equal $[1, 0, \dots, 0]$? How does it depend on X and Z?

5 Best Subset Selection

In this section we describe the Matlab code for best subset algorithm for linear regression. The code is easy straightforward. The new only tricky part is the generation of all possible subsets of a given set of variables. This is done using the `nchoosek` function wich takes two parameters: a vector v of elements to choose from, and the cardinality k of the subsets. The function returs all possible sets of k elements take from v.

```
%evaluare RSS for model with only the intercept and no variables
T = [ones(N,1)];
beta = inv(T'*T)*T'*y;
rss = (y-T*beta)'*(y-T*beta)

pp = 1:p;
for j=1:p-1
    %evaluate best RSS for model with j covariates
    best_rss = -1;
    best_rss_index = 0;
    %list all possible sets of j covariates
    v = nchoosek(pp,j);
    for i=1:length(v)
        %create matrix with the selected covariates
        T = [ones(N,1)];
        %list variables in the set
        for k=1:j
            %add column to the matrix
            T = [T X(:,v(i,k)+1)];
        endfor
        %evaluate beta
        beta = inv(T'*T)*T'*y;
        %evalaute RSS and update if values is smaller the best RSS
        rss = (y-T*beta)'*(y-T*beta);
        if (best_rss == -1)
            best_rss = rss;
            best_rss_index = i;
        else
            if (rss < best_rss)
                best_rss = rss;
                best_rss_index = i;
            endif
        endif
     endfor
     best_rss
     best_rss_index
endfor

%case k=p, all covariates in the model
beta = inv(X'*X)*X'*y;
rss = (y-X*beta)'*(y-X*beta);
```

Exercise: Produce a plot similar to that in Figure 3.5 of [2]

Exercise: Implement Forward-Stepwise and Backward-Stepwise algorithms for linear regression, starting from the code of the Best Subsets described above. Plot the curves associated to the best RSS for each value of k for the two algoritms in the same graph generated in the previous exercise, and compare to those of Best Subsets. Comments the curves you obtained.

Exercise: Plot the curve of $|\hat{\beta}^{\text{best-subset}} - \hat{\beta}^{\text{ols}}|^2$, $|\hat{\beta}^{\text{for-step}} - \hat{\beta}^{\text{ols}}|^2$, $|\hat{\beta}^{\text{back-step}} - \hat{\beta}^{\text{ols}}|^2$ for each value of k in the same graph. Notice that β^{ols} is fixed, does not depends on k, and corresponds to ordinary least squares estimator with p covariates. Comments the curves you obtained.

6 Forward-Stagewise Selection

In this section we present the implementation of Forward-Stagewise selection. As you can see, at each iteration t, only one β_j is updated, differently from previous algorithms.

```
%beta_0 (the intercept) equals mean of y
beta = [mean(y)]

%all other betas associated to the covariates set to zero
beta = [beta ; zeros(p,1)]

do
   %evaluate residual
   residual = y - X*beta;
   v = 0;
   greatestCorrelation = 0;
   %loop over variables
   for i = 2:p+1
      %evaluate covariances and choose the most correlated variable
      c = cov(X(:,i),residual);
      if (i == 2)
          greatestCorrelation = c;
          v = i;
      else if (abs(greatestCorrelation)<abs(c))
            greatestCorrelation = c;
            v = i;
          endif
      endif
   endfor
   greatestCorrelation
   %regress residual on the chosen variable
   b = (X(:,v)'*residual)/(X(:,v)'*X(:,v));
   %update beta
   beta(v) = beta(v) + b;
   beta;
%loop until greatest correlations is under a fixed threshold
until (abs(greatestCorrelation)<0.0001)

%plot beta
beta
```

Exercise: Plot the curve of $|\hat{\beta}^{\text{for-stage}} - \hat{\beta}^{\text{ols}}|^2$ for different values of t. Let β^{ols} be the ordinary least squares estimator with p covariates, which is fixed and does not depends on t. Comments the curve you obtained.

7 Ridge Regression

In this section we present the code for Ridge Regression.

```
%remove first column, since we want to drop the intercept from
%the penalizing term
X(:,1) = [];

%verify that all variable are centered, i.e., the have zero mean
mean(X)

%estimate beta:0 with mean of y
beta_0 = mean(y)
```

```
%test different values of lambda
for lambda=0.0001:0.1:50
   %evaluations of beta coefficients
   beta = inv(X'*X + lambda*diag(ones(p,1)))*X'*y;
endfor
```

We now evaluate the SVD decomposition of X in order to be able to estimated the degrees of freedom associated to λ according to Equation 3.50 of [2].

```
%compute SVD decomposition of X, i.e., X = U*D*V'
[U, D, V] = svd(X);

%evaluate the principal components of X
for i=1:p
    %evaluate variance in different ways, see Equation 3.49 of [2]
    %var(X*V(:,i))
    %var(U(:,i)*D(i,i))
    D(i,i)^2/N
endfor

%in the following, we fix the value of lambda, even if
%we can do that for any lambda
lambda = 0.1;

%evaluate degrees of freedom of lambda
df = 0;
for i=1:p
    df = df + (D(i,i)^2)/(D(i,i)^2+lambda);
endfor
df
```

Exercise: Plot the value of each β_j^{ridge} with respect to the degrees of freedom as in Figure 3.8 of [2]. Comments the curve you obtained.

Question: what is the advantage of ridge regression compared to ordinary least squares, in terms of non-singularities?

Question: Fix $\lambda = 1$. Create a table that contains for each variable u_j the factor of shrinkage of ridge regression. Create a table that contains for each principal component $z_j = Xv_j = u_j d_j$ its variance. How are the variances of normalized principal components u_j and their shrinkage factors related? Why?

8 (Facultative Section) Lars an Lasso

In this section we present the implementation of Lar and the Lasso, as described in [1, Sec 1-3]. Read carefully Section 3.4.2-3.4.3-3.4.4 of [2] for the theory behind Lasso and Lar, and refer to the above mentioned paper for details the implementations of the algorithms.

```
%initialize number of iterations
it = 0;

%this vector is used to save the signs of the correlations
s = ones(p,1);
%value of beta at the first iteration
beta_a = zeros(p,1);
%\hat y at the first iteration
mu_a = X*beta_a;

%this matrix contains the \beta vector at each iteration
B = [zeros(p,1)];
%the t vector contains the l1-norms of \hat \beta at each iteration.
%at the first iteration, the norm is zero, since \beta = 0
t(1) = 0;

%this vector contains the step size, at first iteration is zero
gammmas(1) = 0;

%vector of current selected columns
col = zeros(p,1)

%main loop, cycle until all variables are selected
while(norm(col,1) ~= norm(ones(p,1),1))
    %increase iterations number
    it = it + 1;
    printf('Iteration %d\n',it);

    %evaluate correlations among variables and y-\hat y
    cor = X'*(y-mu_a)
    %initialize the vectors of selected columns
    col = zeros(p,1);

    % find the most correlated variables
    [maxVal maxI] = max(abs(cor));
    for i=1:size(cor,1);
        if (abs(abs(cor(i))-maxVal)<0.0000000001)
            %mark column as selected
            col(i) = 1;
        %save sign of correlation
        s(i) = sign(cor(i));
    else
        col(i) = 0;
    end
end
printf('Selected variables at current iteration');
col'

%build X_a matrix from X
X_a = [];
for i=1:size(col,1);
    if (col(i)==1)
        X_a = [X_a s(i)*X(:,i)]
    end
end

%evaluate G_a
G_a = X_a'*X_a
Ones_a = ones(norm(col,1),1);
M = Ones_a'*inv(G_a)*Ones_a;
%the following code is used to evaluate the square matrix of M by
%diagonalization: let M = V*D*V^{-1}, where D is diagonal, then
%M^{1/2}= V*D^{1/2}*V^{-1}, where D^{1/2} has diagonal elements
%that are the square roots of the diagonal elements of D.
%first compute eigenvalues and eigenvectors of M to obtain
%M = V*D*V^{-1}
[V,D] = eig(M);
%remember that we evaluate M^{-1/2} not just M^{1/2}
A_a = inv(V*sqrt(D)*inv(V));
w_a = A_a*inv(G_a)*Ones_a
u_a = X_a*w_a;

a = X'*u_a;

flag = 0;
for i=1:size(col,1);
    if (col(i)==0)
        flag = flag + 1;
        %see formula 2.13 of [1]
        el1 = (maxVal-cor(i))/(A_a-a(i));
        el2 = (maxVal+cor(i))/(A_a+a(i));
        if (el1>0 && el2>0)
            temp = min(el1,el2);
        end
```

```
            if (el1<=0 && el2>0)
                temp = el2;
            end
            if (el2<=0 && el1>0)
                temp = el1;
            end
            if (el1<0 && el2<0)
                printf('ERROR unexpected condition\n',i);
            end
            %gamma is the step size for the update of mu_a
            if (flag==1)
                gamma = temp;
            else
                gamma = min(gamma,temp);
            end
            end
    end
    %check if this is the last step
    if (col==ones(p,1))
        gamma = maxVal/A_a;
    end

    gamma

    %solve linear system: X beta_aTemp = mu_a, corresponds to
    %linsolve(X,mu_a) in matlab
    beta_aTemp = X \ mu_a;
    beta_aTemp

    %save gamma at the current iteration
    gammas(it+1) = gamma;
    mu_a = mu_a + gamma*u_a;

    %beta at current iteration
    beta_a = X \ mu_a

    %evaluate new value for t
    t(it+1) = norm(abs(beta_a),1)
    %B contains all beta vectors at differet iterations
    B = [B beta_a];
end

B'
gammas'
```

```
printf('Number of iterations %d\n',it);
```

Exercise: Plot the value of the β_j^{lar} coefficients over t, for each iteration of the algorithm, as in Figure 3 (left) of [1].

Exercise: Implement the Lasso modification of Lar, as described in [1] and [2]. Plot the value of the β_j^{lasso} coefficients over t, for each iteration of the algorithm, as in Figure 1 of [1].

Question: Compare the results of Lasso and Lar. Can you see any difference? Comment on the maximum number for steps for both algorithms

References

1. B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.
2. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.