A Octave/Matlab Tutorial for Linear Methods for Classification

Matteo Matteucci

Pattern Analysis and Machine Intelligence, 2012
Politecnico di Milano
Document revision 1.0, June 12, 2012

1 Introduction

This tutorial will guide you though the implementation of some of the linear methods for classification presented in Ch.4 of [1]. The code has been tested with Octave 3.4.0, available at [1], and it is going to be tested in Matlab as well. Feel free to email me for problems or comments about the code. See the documetation about octave at [2]. The *South African Heart Disease* dataset used in this tutorial is the same used in the book, and it can be downloaded from [3]. The file you download from the course website has bee edited to convert the `famhist` attribute coding from Absent/Present into 0/1; keep this in mind if you downloaded the original file.

2 Import the dataset

The first line of file *SAheart.data* contains the name of the covariates and of the output variable. Starting from the second row, the first column contains the number of observations (instances), from column 2 to 10, there are the covariates, and the last column contains the indicator for the class (i.e., presence of a Cardiac Heart Disease).

If you downloaded the dataset from the course website, to load it into Octave it should be enough to execute

```
data = dlmread('SAheart.data',',',1,0)
X = data(:,1:9)
Y = data(:,10)
```

where we split the data matrinx into input features `X` and target class `Y`.

In the dataset there is no distinction between trainig set and test set, but we know tha t a proper evaluation requires the use of two independent datasets. To generate them we use a *Stratified Sampling Procedure*; this is a difficult name for a rather simple procedure to guarantee that the percentage of records per class is respected in both training and testing datasets.

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

```
function [training, testing] = StratifiedSampling(Y,test_ratio)

% split the records in positive and negative examples
rows = 1:length(Y);
pos_idx = rows(Y==0);
neg_idx = rows(Y==1);

% identify the number of records corrsponding to test_ratio
pos_n = floor(length(pos_idx)*test_ratio);
neg_n = floor(length(neg_idx)*test_ratio);

% generate a random permutation for the two sets of indexes
pos_rand = randperm(length(pos_idx))';
neg_rand = randperm(length(neg_idx))';

% extract the given percenttage of positive and negative records
testing = [pos_idx(pos_rand(1:pos_n)) ...
neg_idx(neg_rand(1:neg_n))];
training = [pos_idx(pos_rand(pos_n+1:end)) ...
neg_idx(neg_rand(neg_n+1:end))];

% just an ahestetic sorting
sort(testing);
sort(training);
```

IN THE FOLLOWING TUTORIAL WE ASSUME TO HAVE A 30% OF TESTING DATA, BUT THESE ARE RANDOM, SO WE EXPECT OUR RESULT TO BE “SIMILAR” TO THE BOOK AND NOT EXACTLY EQUAL (THE SAME WILL BE FOR YOUR HOMEWORK RESULTS)

3 Linear regression on the indicator matrix

Following the simple formulas in [Ch.4.2] of [1] we are going to implement the simplest linear classification algorithm. Since the classes are 0/1 building the `Y` indicator matrix can be done quite easily.

```
function beta = linearRegression_train(X,Y)
    %extend the input vector
    X = [ones(size(X, 1), 1) X];

    %build the indicator matrix (ad-hoc for 2 classes)
    Y = [Y not(Y)];

    %computation of coefficients for the 2 models
    beta = inv(X'*X)*X'*Y;
end
```

Exercise: Whats the meaning of the vector of ones we add to the input features? (Don't worry this is just a warm up question!)

Reading from the Octave Manual, for better numerical stability with sparse matrixes and badly conditioned regression problems you should use $Y = A\backslash B$, rather than $Y = inv(A) * B$

```
function beta = linearRegression_train(X,Y)
    %extend the input vector
    X = [ones(size(X, 1), 1) X];

    %build the indicator matrix (ad-hoc for 2 classes)
    Y = [Y not(Y)];

    %computation of coefficients for the 2 models
    beta = (X'*X)\X' * Y;
end
```

With the `beta` vector estimated from the data it is possible to classify new instances applying linear regression to them as in the following function.

```
function Y = linearRegression_test(X,beta)
%extend the input vector
    X = [ones(size(X, 1), 1) X];

    %evaluate the output of the two models
    Y_hat = X*beta;

    %select the one with the highest response
    Y = (Y_hat(:,1)>Y_hat(:,2));
end
```

You can check the result by training on the training set and testing on the testing set

```
beta = linearRegression_train(X(training,:),Y(training))
Y_predicted = linearRegression_test(X(testing,:),beta)
```

Exercise: What are the training and testing error (percentage of wrong class predictions) in this case? Does this result surprise you? Why?

4 Linear discriminant analysis

Now move to the implementation of Linear Discriminant Analysis with and without Fisher projection. Let start from the simple one; in this case we need to split the dataset according to the class and compute means and priors for each of them. The common covariance comes from the pooling of the two variances.

```
function [mu_0, mu_1, sigma, pi_0, pi_1] = ...
linearDiscriminantAnalysis_train(X, Y)
    % indexes of samples for the two classes
    first_class = find(Y == 1);
    second_class = find(Y == 0);

    % number of samples
    N = size(X, 1);

    % a priori probabilities of the 2 classes
    pi_1 = size(X(first_class, :), 1) / N;
    pi_0 = size(X(second_class, :), 1) / N;

    % centroids of the 2 classes
    mu_1 = mean(X(first_class, :), 1);
    mu_0 = mean(X(second_class, :), 1);

    % common pooled covariance matrix
    % (it turn out we can summ the covariances)
    sigma = zeros(size(X, 2));
    for i=1:size(first_class, 1)
        Xi = X(first_class(i), :);
        sigma = sigma + (Xi - mu_1)' * (Xi - mu_1);
    end
    for i=1:size(second_class, 1)
        Xi = X(second_class(i), :);
        sigma = sigma + (Xi - mu_0)' * (Xi - mu_0);
    end
    sigma = sigma / N;
```

With the mu_0, mu_1, sigma, pi_0, pi_1 parameters estimated from the data it is possible to classify new instances applying linear discriminant analysis to them as in the following function.

```
function Y = linearDiscriminantAnalysis_test(X, mu_0, mu_1, ...
sigma, pi_0, pi_1)
    sigmaInv = inv(sigma);

    % comparing the discriminants is a simplification of
    % comparing the posterior probabilities
    discriminant_1 = X*sigmaInv*mu_1' - ...
     0.5*mu_1*sigmaInv*mu_1' + log(pi_1);
    discriminant_0 = X*sigmaInv*mu_0' - ...
     0.5*mu_0*sigmaInv*mu_0' + log(pi_0);
    Y = (discriminant_1>discriminant_0);
end
```

Exercise: What are the training and testing error (percentage of wrong class predictions) in this case? Does this result surprise you? Why?

Lets now introduce the Fisher projection in the linear discriminant analysis; the code notation follow the notation from [Ch.4.3.3] of [1] so you should be able to follow it with the textbook on your side (do that!)

```
function a = FisherProjection(X, Y)

    % we assume to know that we have 2  classes
    first_class = find(Y == 1);
    second_class = find(Y == 0);
    N = size(X, 1);

    % centroids of the two classes
    mu_1 = mean(X(first_class, :), 1);
    mu_0 = mean(X(second_class, :), 1);
    M = [mu_1; mu_0];

    % common pooled within-class covariance matrix
    W = zeros(size(X, 2));
    for i=1:size(first_class, 1)
        Xi = X(first_class(i), :);
        W = W + (Xi - mu_1)' * (Xi - mu_1);
    end
    for i=1:size(second_class, 1)
        Xi = X(second_class(i), :);
        W = W + (Xi - mu_0)' * (Xi - mu_0);
    end
    W = W / N;

    % maximization of a'*B*a / a'*w*a via SVD
    [Vw, Dw, Vw] = svd(W);
    Whalf = Vw * Dw ^ (1/2) * Vw'; % Whalf'*Whalf == W

    % we substitute b = Whalf * a as the independent variable
    Wminushalf = inv(Whalf);
    Mstar = M*Wminushalf;
    for i=1:size(M,1)
        Mstar(i,:) = Mstar(i,:)-mean(Mstar);
    end
    Bstar = Mstar'*Mstar;
    [Vstar, Db, Vstar] = svd(Bstar);

    % we are interested in the eigen vectors of Bstar
    a = Wminushalf * Vstar;
end
```

Once the Fisher projection has been computed it is possible to use it for computing the linead discriminat classifier after projecting the data on a reduced subspace

```
a = FisherProjection(X,Y)
reducedX = X*a(:,1);
[mu_0, mu_1, sigma, p_0, p_1] = linearDiscriminantAnalysis_train(reducedX, Y)
```

Exercise: What are the training and testing error (percentage of wrong class predictions) in this case? Does this result surprise you? Why?
Exercise: What happens if we decide to use a number of dimensions K = 2 for the Fisher projection? Do the numbers differ?
Exercise: To understand what is happening plot the points in the transformed space with K=1 (one dimentional plot) and K=2 (two dimensional plot) with a different color for each class. As for the Figures ... in the book generate the plot with different principal components.

5 Quadratic discriminant analysis

Here we are going to implement two different methods for non linear discriminant analysis. The first one is the quadratic version of the linear regression on the indicator matrix, while the second one implements Quadratic Discriminant Analysis in the “right” way.

The only tricky part in computing the quadratic variant of the linear regression on the indicator matrix is the generation of the covariates in the quadratic space

```
function extendedX = expandToQuadraticSpace(X)
    % adds new columns to extendedX; keeps X for other calculations
    extendedX = X;
    for i=1:size(X, 2)
        for j=1:size(X, 2)
            newColumn = X(:, i) .* X(:, j);
            extendedX = [extendedX newColumn];
        end
    end
end
```

Once this expansion has been obtained it is possible to run directly the linear regression method we have seen at the beginning of this tutorial:

```
quadX = expandToQuadraticSpace(X);

%check this out!
size(quadX)

beta = linearRegression_train(quadX, Y);
```

Exercise: Have you tried this with the numerically stable version of linear regression?

Anyway the trademarked version of Quadratic Discriminant Analysis is a different one; I am talking about the linear discriminant approach with "unpooled variance". Its is not much different from the linear Discriminat case right?

```
function [mu_0, mu_1, sigma_0, sigma_1, pi_0, pi_1]= ...
quadraticDiscriminantAnalysis_train(X, Y)
    % indexes of samples for the two classes
    first_class = find(Y == 1);
    second_class = find(Y == 0);
    % number of samples
    N = size(X, 1);

    % a priori probabilities of the 2 classes
    pi_1 = size(X(first_class, :), 1) / N;
    pi_0 = size(X(second_class, :), 1) / N;

    % centroids of the 2 classes
    mu_1 = mean(X(first_class, :), 1);
    mu_0 = mean(X(second_class, :), 1);

    % covariance matrices for each class (2 classes)
    % the unbiased estimator has N - 1 at the denominator
    sigma_1 = zeros(size(X, 2));
    for i=1:size(first_class, 1)
        Xi = X(first_class(i), :);
        sigma_1 = sigma_1 + (Xi - mu_1)' * (Xi - mu_1);
    end
    sigma_1 = sigma_1 / (size(first_class, 1));
    sigma_0 = zeros(size(X, 2));
    for i=1:size(second_class, 1)
        Xi = X(second_class(i), :);
        sigma_0 = sigma_0 + (Xi - mu_0)' * (Xi - mu_0);
    end
    sigma_0 = sigma_0 / (size(second_class, 1));
end
```

At run time to use this classifier you just need to compare the non linear discriminants

```
function Y = quadraticDiscriminantAnalysis_test(X, mu_0, mu_1, ...
 sigma_0, sigma_1, pi_0, pi_1)
    sigma_1_inv = inv(sigma_1);
    sigma_0_inv = inv(sigma_0);
    discriminant_1 = zeros(size(X, 1), 1);
    discriminant_0 = zeros(size(X, 1), 1);

    % for each single sample i, compute discriminants value;
    % there is no single expression like for LDA
    for i=1:size(X, 1)
        x = X(i, :);
        discriminant_1(i) = - 0.5*log(det(sigma_1)) - ...
         0.5*(x-mu_1)*sigma_1_inv*(x-mu_1)' + log(pi_1);
    end
    for i=1:size(X, 1)
        x = X(i, :);
        discriminant_0(i) = - 0.5*log(det(sigma_0)) - ...
         0.5*(x-mu_0)*sigma_0_inv*(x-mu_0)' + log(pi_0);
    end

    % producing a single column Y of classifications
    % via matrices operations
    Y = (discriminant_1 > discriminant_0);
end
```

Exercise: What are the training and testing error (percentage of wrong class predictions) in **these** cases? Does this result surprise you? Why?

6 Logistic regression

Last but not least we got to Logistic Regression as described in [Ch.4.4] of [1]. Following the book its implementation is quite easy ... in the binary case!

```
function beta=logisticRegression_train(X, Y, iterations)
```

```
    % adding columns of 1 to allow for an intercept in the model
    X1 = [ones(size(X, 1), 1) X];

    % suggest init using zeros
    beta=zeros(size(X1,2),1);

    for i=1:iterations
        P = exp(X1*beta)./(1 + exp(X1*beta));
        %loglikelihood should increase at each iteration
        loglikelihood = Y'*log(P)+(1-Y)'*(log(1-P))
        W = diag(P.*(1-P));
        Z = X1*beta + W\(Y-P);
        beta = (X1'*W*X1)\(X1'*W) * Z;
    end
end
```

Exercise: Does the log likelihood increase as it should be?

For the testing you need

```
function Y = logisticRegression_test(X, beta)
     % adding columns of 1 to allow for an intercept in the model
     X1 = [ones(size(X, 1), 1) X];

     % compute probability of class 1 give beta
     P = exp(X1*beta)./(1+ exp(X1*beta));
     Y = (P>0.5);
end
```

Exercise: What are the training and testing error (percentage of wrong class predictions) in these cases? Does this result surprise you? Why?

7 (Facultative Section) Multiclass Dataset

Have done all this stuff for two classes you might be wondering if this all works for a number of classes higher than 2. The answer is YES! Interested students, with plenty of time, might decide to modify the code to have it running for a generic number of classes coded as 1, 2, 3 ... K and test your work in the *Vowel Dataset* used in the book (available for downloaded from [4]).

References

1. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[4] http://www-stat.stanford.edu/ tibs/ElemStatLearn/