Linear regression

1 Tutorial

This tutorial explains how to perform linear regression in MATLAB using the statistics toolbox (the tutorial was tested on the release R2012b).

1.1 Simple linear regression

Linear regression is a method for predicting the value of an output Y from an input X having multiple attributes or *features*. Simple regression refers to the case where the input is represented as a scalar value. For example, let us consider the problem of predicting the height of a tree from its diameter. The data below are measurements of height (in m) and diameter d (in cm) of 18 Corsican pines:

```
x= [32 31 30 29 29 28 25 23 20 18 17 17 16 16 15 13 11 11]';
y= [22.7 22.7 22.6 22.6 21.9 21.9 21.8 21.0 20.4 18.6 ...
    19.2 18.9 18.5 18.1 17.7 17.2 16.5 15.5]';
```

We can display the data using `plot(x,y,'o')`. Is there some correlation between the two quantities? If so, let us try to understand how trees grow!!!

We can explore the relationship between the variables using linear regression. This amounts to modeling the prediction as a straight line $f_w(X) = w_1 + w_2X$, whose parameters w_1, w_2 are computed as follows:

```
function w = simple_regression(y, x)
% x = sample inputs
% y = sample outputs
% w = parameters of the linear prediction f(x) = w1 + w2 * x

mx = mean( x(:) );
my = mean( y(:) );

w(2) = sum((x(:)-mx) .* (y(:)-my)) / sum((x(:)-mx).^2);
w(1) = my - w(2) * mx;
```

We can estimate the parameters from the given data:

```
w = simple_regression(y,x);
z_lin = w(1) + w(2) * x;
```

and then visualize how the prediction fits the data:

```
figure;
subplot(2,2,1);
plot(x, y, 'o', x, z_lin, 'r-');  title('Linear prediction');

subplot(2,2,2);
plot(z_lin,y-z_lin,'x', [min(z_lin) max(z_lin)],[0 0],'r-');
title('Residual plot');
```

Looking at the residual plot, we can see that the distribution of residuals is U-shaped, indicating that the prediction could be better represented by a nonlinear model. In particular, the data seems to suggest a logarithmic relationship:

```
% logarithmic prediction
w = simple_regression( y, log(x) );
z_log = w(1) + w(2) * log(x);

% visualization
subplot(2,2,3);
plot(x, y, 'o', x, z_log, 'r-'); title('Logarithmic prediction');
subplot(2,2,4);
plot(z_log,y-z_log,'x', [min(z_log) max(z_log)],[0 0],'r-');
title('Residual plot');
```

The fitting now looks definitely better!

1.2 Multiple linear regression

In general, we can learn a better prediction by taking into account several pieces of information. For example, the fuel efficiency of a car is more likely to be correlated with several factors, such as its weight, horsepower, number of cylinders, manufacture year, and country of origin. The statistics toolbox includes a dataset of cars manufactured in 1970 − 1982. We can use the function `plotmatrix` to plot the "miles per gallon" versus the other measurements. Before doing so, we need to handle the categorical variables (Cylinders, Model_Year and Origin) by using the commands `ordinal` and `nominal`.

```
load('carbig');

% handle the categorical variables
Cylinders  = ordinal(Cylinders);
Model_Year = ordinal(Model_Year);
Origin     = nominal(Origin);

% visualize data
figure;
plotmatrix( [Weight Horsepower double(Cylinders)...
                double(Model_Year) double(Origin)], MPG);
```

In this example, we use a prediction model based on an intercept (i.e., a vector of ones), the weight and horsepower features together with their square values (due to their nonlinear relationship with the mileage), and the binary representation of the three categorical features (computed with the function `dummyvar`).

```
% output
y  = MPG;

% inputs
x1 = ones(size(MPG));
x2 = [Weight, Weight.^2];
x3 = [Horsepower, Horsepower.^2];
x4 = dummyvar(Cylinders);
x5 = dummyvar(Model_Year);
x6 = dummyvar(Origin);
```

Now, we can use the function `regress` to perform the linear regression.

```
x = [x1, x2, x3, x4, x5, x6];
w = regress(y, x);
```

Note that the function `regress` treats NaNs in `x` or `y` as missing values, and thus it removes them before performing the regression. Consequently, if our data contains NaNs, we need to remove them before proceeding.

```
wasnan = isnan(y) | any(isnan(x),2);
y(wasnan)   = [];
x(wasnan,:) = [];
```

The prediction and the residuals can be computed as

```
z = x*w;    % predictions
r = y - z;  % residuals
```

We can asses the quality of fitting by computing the MSE and R^2 statistics, as well as showing the residual plot.

```
% statistics
MSE = mean( r.^2 );
R2  = 1 - MSE / mean( (y-mean(y)).^2 );
text = sprintf('R^2 = %2.3f -- MSE = %2.3f', R2, MSE);

% residual plot
plot(z, r, 'x', [min(z) max(z)], [0 0], 'r-');
title(text);
```

The residuals are randomly distributed around the zero, which is one of the indicators of a good prediction.

1.3 Regularized regression

The statistics toolbox supports both ridge and lasso regression. Let us consider the toy example of polynomial fitting.

```
% define a polynomial
P = @(t) ...
[ones(size(t,1),1) t.^2 t.^3 t.^4 t.^5 t.^6 t.^7 t.^8 t.^9];

% generate data
t = (-1 : 0.2 : 1)';
x = P(t);
y = randn( length(t), 1 );
```

We can perform a regularized regression with the functions `ridge` and `lasso`. Unlike the function `regress`, the matrix `x` does not have to include a column of ones, because the data are standardized before performing the regression. Also, an additional input specifying the regularization strength is required.

```
% standard regression
w1 = regress(y, x);

% ridge regression (RECALL: no column of ones in x !!!)
lambda = 0.05;
w2 = ridge(y, x(:,2:end), lambda, 0);

% lasso regression (RECALL: no column of ones in x !!!)
lambda = 0.05;
[w3,infos] = lasso( x(:,2:end), y, 'Lambda', lambda);
w3 = [infos.Intercept; w3];
```

The prediction can be visualized as follows:

```
s = (-1 : 0.001 : 1)';
plot(t,y,'ok',s,P(s)*w1,'b', s,P(s)*w2, 'r', s,P(s)*w3,'g');
legend({'Samples' 'Standard' 'Ridge' 'Lasso'})

fprintf('Ridge: %d/%d\n', sum(w2~=0), length(w2) );
fprintf('Lasso: %d/%d\n', sum(w3~=0), length(w3) );
```

In this example, ridge and lasso predictions lead to a similar prediction, although the latter shrinks some parameters to zero, meaning that only a subset of features contributes to the prediction.

Note also that both functions accept a vector of values for controlling the regularization. This is useful to visualize how the estimated parameters evolve with respect to the regularization strength.

```
[w,infos] = lasso(x(:,2:end),y, 'Lambda', 0:0.01:1 );
lassoPlot(w, infos, 'PlotType', 'Lambda','XScale','log');
```

1.4 Cross-validation

Let us consider a more realistic example. We want to predict the diabetes progression (after one year) from the current status of a patience evaluated in terms of age, sex, body mass index, average blood pressure, and six blood serum measurements. To do so, we can load the file `diabetes.mat`, transform the “sex” feature into a 0-1 variable, and add a column of ones.

```
load('diabetes'); % x = matrix of inputs, y = vector of outputs

x(:,2) = double( x(:,2)==1 ); % convert to a 0-1 variable
x = x2fx(x, 'linear');        % add column of ones
```

Our objective is to compare the different types of linear regression using cross validation. To this end, we can use the command `cvpartition` to define $K = 10$ different random partitions on a dataset of a specified size.

```
rng('default');
cv = cvpartition( size(x,1), 'kfold', 10 );
```

The following script illustrates how to cross-validate the MSE obtained with the ordinary linear regression.

```
MSE_train = 0;
MSE_valid = 0;
for i = 1 : cv.NumTestSets
    % training data
    idx_train = cv.training(i);
    x_train = x(idx_train,:);
    y_train = y(idx_train,:);

    % validation data
    idx_valid = cv.test(i);
    x_valid = x(idx_valid,:);
    y_valid = y(idx_valid,:);

    % linear regression
    w = regress(y_train, x_train);

    % training MSE
    r = y_train - x_train * w;
    MSE_train = MSE_train + mean(r.^2);

    % validation MSE
    r = y_valid - x_valid * w;
    MSE_valid = MSE_valid + mean(r.^2);
end
MSE_train = MSE_train / cv.NumTestSets;
MSE_valid = MSE_valid / cv.NumTestSets;
```

Note that the above loop is already implemented in the function `crossval`, hence there is no need to manually code it once you have understood how cross-validation works. This is the command for reproducing the above algorithm:

```
MSE_regress = crossval(@regress_mse, y, x, 'partition', cv);
MSE_regress = mean(MSE_regress);
```

The procedure `crossval` takes a "function handler" as a first argument. This handler is called repeatedly on a different partition of the arrays y and x passed as second and third arguments, as follows:

```
out = fun(y_train, x_train, y_valid, x_valid)
```

Therefore, the function `fun` should use `y_train` and `x_train` to learn a prediction model, and then return some criterion computed on `y_valid` and `x_valid` using that learned prediction. For example, this is the function that computes the validation MSE relative to the ordinary linear regression.

```
function MSE = regress_mse(y_train,x_train, y_valid,x_valid)

% linear regression
w = regress(y_train, x_train);

% validation MSE
r = y_valid - x_valid * w;
MSE = mean( r.^2 );
```

The functions associated with ridge and lasso regression are the following. They have an additional argument meant for cross-validating the regularization.

```
function MSE = ridge_mse(y_train,x_train,y_valid,x_valid, L)

  for i = 1 : length(L)
      w = ridge( y_train, x_train(:,2:end), L(i), 0 );

      r = y_valid - x_valid * w;
      MSE(i) = mean( r.^2 );      % validation MSE
  end



function MSE = lasso_mse(y_train,x_train,y_valid,x_valid, L)

  for i = 1 : length(L)
      [w,c] = lasso(x_train(:,2:end),y_train,'Lambda',L(i));
      w = [c.Intercept; w];

      r = y_valid - x_valid * w;
      MSE(i) = mean( r.^2 );      % validation MSE
  end
```

So doing, we can cross-validate both ridge and lasso regression for different levels of regularization, and compare them with the ordinary regression:

```
% cross-validation
MSE_regress = crossval(@regress_mse, y, x, 'partition', cv);

% cross-validation with a parameter
lambda = 0 : 0.2 : 1.6;
MSE_ridge=crossval( @(a,b,c,d) ridge_mse(a,b,c,d,lambda),...
                      y, x, 'partition', cv );
MSE_lasso=crossval( @(a,b,c,d) lasso_mse(a,b,c,d,lambda),...
                      y, x, 'partition', cv );

% average the observations
MSE_regress = mean(MSE_regress);
MSE_ridge   = mean(MSE_ridge);
MSE_lasso   = mean(MSE_lasso);

figure;
plot(lambda, ones(size(lambda))*MSE_regress, 'k-');
hold on;
plot(lambda, MSE_ridge, 'b-');
plot(lambda, MSE_lasso, 'r-');
legend({'Ordinary' 'Ridge' 'Lasso'});
```

1.5 Learning curves

Learning curves can be generated similarly. We define the function:

```
function MSE = curves_mse(y_train,x_train, y_valid,x_valid)

  for j = 1 : size(y_train,1);
     w = regress( y_train(1:j,:), x_train(1:j,:) );

     r = y_train(1:j,:) - x_train(1:j,:) * w;
     MSE(1,j,1) = mean( r.^2 );    % training MSE

     r = y_valid - x_valid * w;
     MSE(1,j,2) = mean( r.^2 );    % validation MSE
  end
```

and then we call the function `crossval` as follows:

```
MSE = crossval(@curves_mse, y, x, 'partition', cv);
MSE = mean(MSE);
MSE = reshape(MSE, [length(MSE)/2 2]);
```

Make sure that the datasets used for training and validation have the same size among the different partitions, otherwise you will get an error.

2 Assignment

Flu epidemics constitute a major public health concern, causing respiratory illnesses, hospitalizations, and deaths. According to the National Vital Statistics Reports published in October 2012, influenza ranked as the eighth leading cause of death in 2011 in the United States. Each year, 250′000 to 500′000 deaths are attributed to influenza related diseases throughout the world.

The U.S. Centers for Disease Control and Prevention (CDC) and the European Influenza Surveillance Scheme (EISS) detect influenza activity through virologic and clinical data, including Influenza-like Illness (ILI) physician visits. National and regional data, however, are published with a 1-2 week lag. The Google Flu Trends project was initiated to see if faster reporting can be made possible by considering flu-related online search queries, which are data available almost immediately.

We would like to estimate influenza-like illness (ILI) activity using Google web search logs. Fortunately, one can easily access this data online. Indeed, the CDC publishes on its website the official regional and state-level percentage of patient visits to healthcare providers for ILI purposes on a weekly basis. In addition, Google Trends allows public retrieval of weekly counts for every query searched by users around the world. For each location, the counts are normalized by dividing the count for each query in a particular week by the total number of online search queries submitted in that location during the week. Then, the values are adjusted to be between 0 and 1.

2.1 Understanding the data

The csv file `FluTrain.csv` aggregates this data from the 1st January 2004 to 31st December 2011. In MATLAB, you can load the file as follows

```
T = readtable('FluTrain.csv');
Train.Weeks   = T{:,1};
Train.ILI     = T{:,2};
Train.Queries = T{:,3};
```

where `Weeks` contains the range of dates associated with the observations, `ILI` contains the percentage of ILI-related physician visits for the weeks, and `Queries` contains the fraction of queries that are ILI-related for the weeks, adjusted to be between 0 and 1 (higher values correspond to more search queries).

Question 2.1.1 Before applying analytics tools on the training set, we first need to understand the data at hand. Plot the vector `ILI` as a function of `Queries`. Looking at the time period 2004-2011, which weeks correspond to:

1. the highest percentage of ILI-related physician visits?
2. the highest percentage of ILI-related query fraction?

Select the day corresponding to the start of these weeks.

Question 2.1.2 Let us now understand the data at an aggregate level. Plot the histogram of the vector `ILI` with the MATLAB function `hist`. What best describes the distribution of ILI values?

- ◯ Most of the ILI values are small, with a relatively small number of much larger values (in statistics, this sort of data is called "skew right").
- ◯ The ILI values are balanced, with equal numbers of unusually large and unusually small values.
- ◯ Most of the ILI values are large, with a relatively small number of much smaller values (in statistics, this sort of data is called "skew left").

2.2 Linear regression

When handling a skewed dependent variable, it is often useful to predict the logarithm of the output instead of the output itself. This prevents the small number of unusually large or small observations from having an undue influence on the sum of squared errors of predictive models. In this problem, we will predict the natural log of the ILI value.

Question 2.2.3 Plot the natural logarithm of `ILI` as a function of `Queries`. What does the plot suggest?

- ◯ There is a negative, linear relationship between `log(ILI)` and `Queries`.
- ◯ There is no apparent linear relationship between `log(ILI)` and `Queries`.
- ◯ There is a positive, linear relationship between `log(ILI)` and `Queries`.

Question 2.2.4 Based on our understanding of the data from Question 2.2.3, which model best describes the relation between the values of ILI and Queries?

- ◯ ILI $= \beta_0 + \beta_1 \times$ Queries, where the slope β_1 is negative.
- ◯ ILI $= \beta_0 + \beta_1 \times$ Queries, where the slope β_1 is positive.
- ◯ Queries $= \beta_0 + \beta_1 \times$ ILI, where the slope β_1 is negative.
- ◯ Queries $= \beta_0 + \beta_1 \times$ ILI, where the slope β_1 is positive.
- ◯ log(ILI) $= \beta_0 + \beta_1 \times$ Queries, where the slope β_1 is negative.
- ◯ log(ILI) $= \beta_0 + \beta_1 \times$ Queries, where the slope β_1 is positive.
- ◯ Queries $= \beta_0 + \beta_1 \times$ log(ILI), where the slope β_1 is negative.
- ◯ Queries $= \beta_0 + \beta_1 \times$ log(ILI), where the slope β_1 is positive.

Question 2.2.5 Let's train the regression model from Question 2.2.4 and call it `FluTrend1`. What are the MSE and R^2 for this model on the train set?

2.3 Performance on the test set

Load the csv file `FluTest.csv` containing the 2012 weekly data of the ILI-related search queries and the observed percentage of ILI-related physician visits.

```
T = readtable('FluTest.csv');
Test.Weeks   = T{:,1};
Test.ILI     = T{:,2};
Test.Queries = T{:,3};
```

Normally, we would obtain the predictions from the model `FluTrend1` as

```
Pred_logILI = FluTrend1(1) + FluTrend1(2) * Test.Queries;
```

However, since we trained our model with `log(ILI)`, the vector `Pred_logILI` would actually contain predictions of the logarithmic value of ILI. We are instead interested in obtaining predictions of the ILI value itself. We can convert from predictions of log(ILI) to predictions of ILI via exponentiation (function `exp`).

Question 2.3.6 What is our prediction for the ILI value corresponding to the week of 11th March 2012?

Question 2.3.7 What is the square error between the predicted and the observed value for the week of March 11, 2012?

Question 2.3.8 What is the MSE of our prediction on the test set?

2.4 Training a time series

The observations in this dataset are consecutive weekly measurements of ILI and Queries. This sort of dataset is called a *time series*. Often, statistical models can be improved by predicting the current value of the output using the value of the output from earlier weeks. In our models, this means we will predict the ILI value in the current week using the ILI values predicted in previous weeks.

First, we need to decide the amount of time to lag the observations. Because the ILI variable is reported with a 1 or 2 week lag, a decision maker cannot rely on the previous week's ILI value to predict the current week's value. Instead, the decision maker will only have data available from 2 or more weeks ago. We will build a new vector called `ILI_Lag2` that contains the ILI value from 2 weeks before the current observation. In MATLAB, this can be done as follows

```
Train.ILI_Lag2 = [NaN; NaN; Train.ILI(1:end-2)];
```

The NaNs indicate that we have no previous predictions in the first two weeks.

Question 2.4.9 Plot the log of `ILI_Lag2` against the log of `ILI`. Which best describes the relationship between these two variables?

- ◯ There is a strong negative relationship between `log(ILI_Lag2)` and `log(ILI)`.
- ◯ This is a weak or no relationship between `log(ILI_Lag2)` and `log(ILI)`.
- ◯ There is a strong positive relationship between `log(ILI_Lag2)` and `log(ILI)`.

Question 2.4.10 Using the training set, train a linear regression model for predicting `log(ILI)` using `Queries` and `log(ILI_Lag2)`. Call this model `FluTrend2`. What is the MSE and R^2 for this model on the training set?

Question 2.4.11 Compare the MSE and R^2 statistics obtained with the models `FluTrend1` and `FluTrend2` on the training set, and plot the residual plot for both models. Which statement is the most accurate?

- ◯ `FluTrend2` is about the same quality as `FluTrend1` on the training set.
- ◯ `FluTrend2` is a stronger model than `FluTrend1` on the training set.
- ◯ Due to overfitting, `FluTrend2` is a weaker model then `FluTrend1` on the training set.

2.5 Evaluating the time series on the test set

So far, we have created the vector `Train.ILI_Lag2` in the training set. To assess the prediction quality of our `FluTrend2` model, we also need to create a *different* vector `Test.ILI_Lag2` in the test data. To do so, note that the training and testing sets were split sequentially. Therefore, the training set `FluTest.csv` contains the observations from 2004-2011, and the test set `FluTest.csv` contains all observations from 2012. There is no time gap between the two datasets, meaning the first observation in the testing data was recorded one week after the last observation in the training data. From this, we can identify how to fill in the missing values of the vector `Test.ILI_Lag2` in the test set.

Question 2.5.12 Which value should be used to fill in the first position of the vector `Test.ILI_Lag2` in the test set?

- ◯ The ILI value of the second-to-last observation in the training data.
- ◯ The ILI value of the last observation in the training data.
- ◯ The ILI value of the first observation in the training data.
- ◯ The ILI value of the second observation in the training data.

Question 2.5.13 Which value should be used to fill in the second position of the vector `Test.ILI_Lag2` in the test set?

- ◯ The ILI value of the second-to-last observation in the training data.
- ◯ The ILI value of the last observation in the training data.
- ◯ The ILI value of the first observation in the training data.
- ◯ The ILI value of the second observation in the training data.

Question 2.5.14 Fill in the two missing values into the vector `Test.ILI_Lag2` using the answer to the previous questions. What are these new two values?

Question 2.5.15 On the test set, predict the ILI values using the `FluTrend2` model, again remembering to call the `exp` function to obtain predictions for `ILI` instead of `log(ILI)`. What is the MSE for this model on the test set?

Question 2.5.16 Which model obtained the best MSE on the test set?

- ◯ `FluTrend1`
- ◯ `FluTrend2`

2.6 Further study

In this problem, we used a simple time series model with a single lag term. ARIMA models are a more general form of the model we built, which can include multiple lag terms, as well as more complicated combinations of previous values of the dependent variable. If you are interested in learning more, check out ARIMA or the available online tutorials for these sorts of models.