Testing the Data

Linear Regression

The function `glmfit` stands for generalised linear model fit and can be used for linear regression and log regression as well as a few more (binomial etc).

For linear regression `regress` is also a good place to start.

```
x=1:10
```

```
x =
     1     2     3     4     5     6     7     8     9    10
```

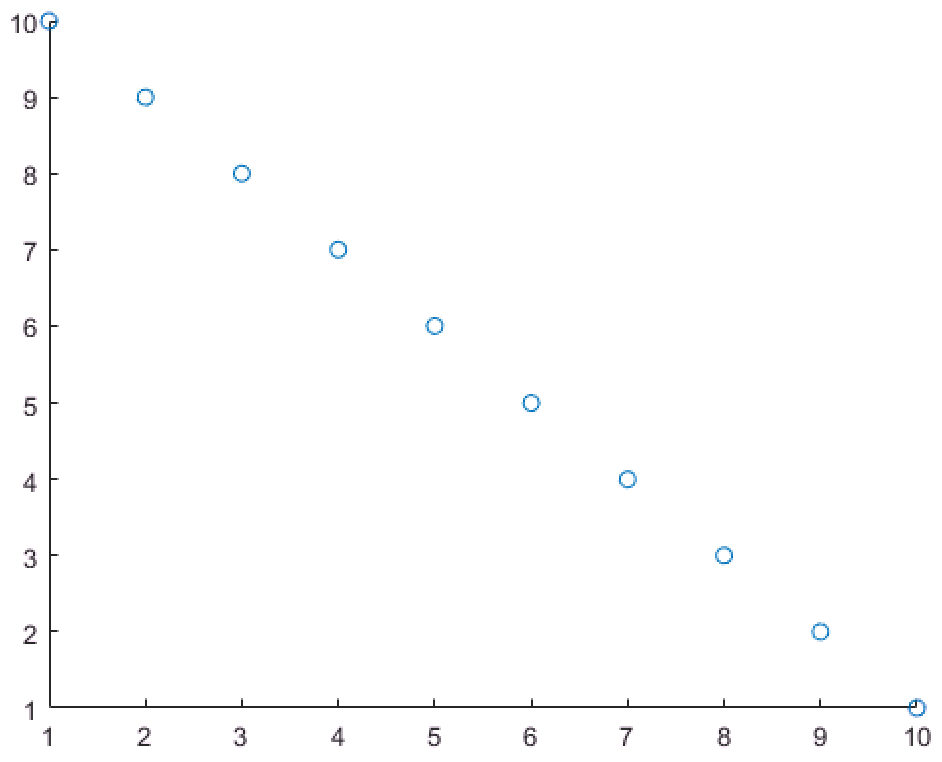
•

```
y=10:-1:1
```

```
y =
    10     9     8     7     6     5     4     3     2     1
```

•

```
scatter(x,y)
```



```
corr(x',y')
```

```
ans = -1
```

```
regress(x',y')
```

```
ans = 0.5714
```

```
y/x
```

```
ans = 0.5714
```

```
```

Compare to `glmfit` to get a slope and intercept

```
scatter(BourkeN,BourkeS)

linear_coeff= glmfit(BourkeN, BourkeS)
```

```
linear_coeff =
    8.6732
    1.0535
```

•

You can also access test statistics:

```
[linear_coeff, ~, stats]= glmfit(BourkeN, BourkeS)
```

```
linear_coeff =
    8.6732
    1.0535
```

•

```
stats = struct with fields:
         beta: [2×1 double]
          dfe: 47948
         sfit: 365.8275
            s: 365.8275
      estdisp: 1
         covb: [2×2 double]
           se: [2×1 double]
    coeffcorr: [2×2 double]
            t: [2×1 double]
            p: [2×1 double]
        resid: [47950×1 double]
       residp: [47950×1 double]
       residd: [47950×1 double]
       resida: [47950×1 double]
          wts: [47950×1 double]
```

```
stats.se
```

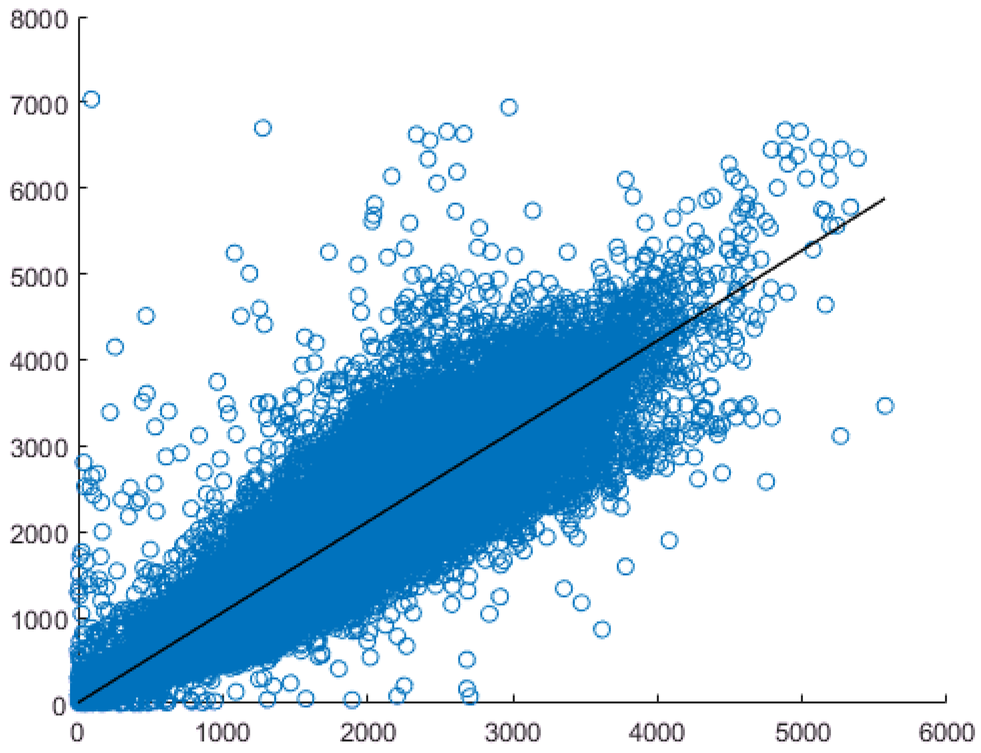
```
ans =
    2.2884
    0.0015
```

-

```
y_linmodel= linear_coeff(1)+ linear_coeff(2).*BourkeN;
hold on; plot(BourkeN,y_linmodel, 'r')
```

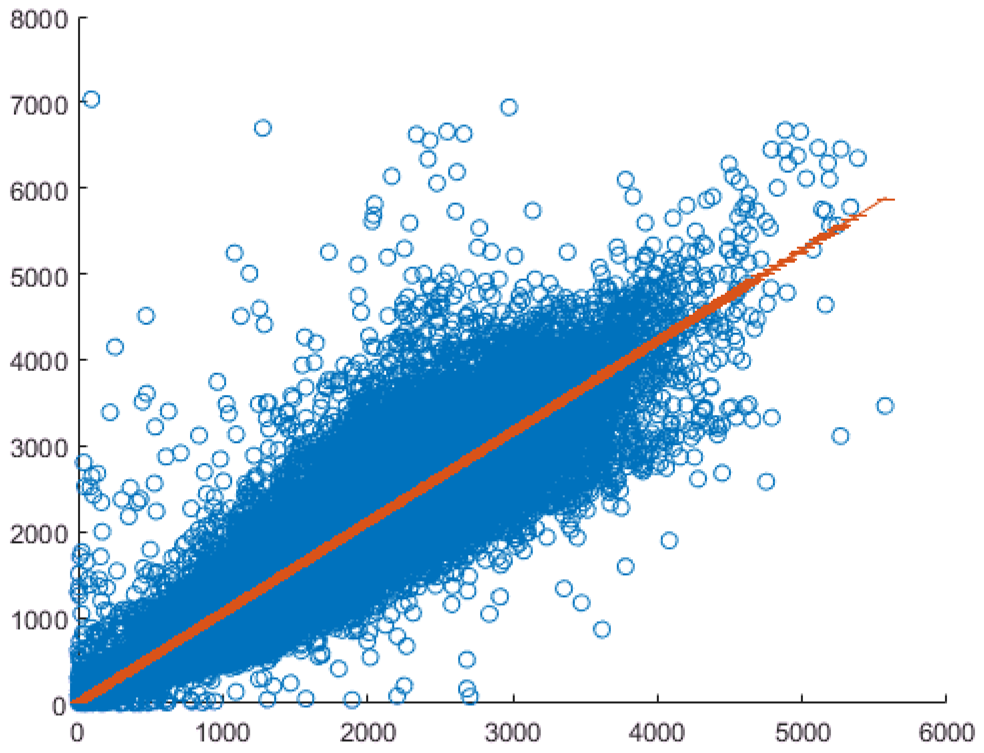
Challenge

```
%% CHALLENGE 1
% Get regress to output upper and lower bounds
% on the estimate
% ie [coeff,bounds] = regress()

%% CHALLENGE 2
% Use glmfit to output confidence limits and
% plot them as dashed lines on your plot
% HINT: use the standard error (se) in the output
% statistics of glmfit
SE= stats.se; % stats is a structure, use '.' notation to get standard error along
interceptbounds= [linear_coeff(1)-2*SE(1), linear_coeff(1)+2*SE(1)]; % set intercept bounds as
y_lowerlimit=interceptbounds(1)+linear_coeff(2).*BourkeN; % Make the lower limit equation
y_upperlimit=interceptbounds(2)+linear_coeff(2).*BourkeN; % Make the upper limit equation

% plot the equations
plot(BourkeN, y_lowerlimit, 'k') %'k' makes the lines black
plot(BourkeN, y_upperlimit, 'k')
```



```
% What other stats can you get from glmfit?
% Alternate method using errorbars
figure;
scatter(BourkeN,BourkeS); hold on
```

```
errorbar(BourkeN, y_linmodel, ones(size(BourkeN))*2*SE(1))
```



There is also a function (I think it is for people who like Stata) that will print statistics into the command window

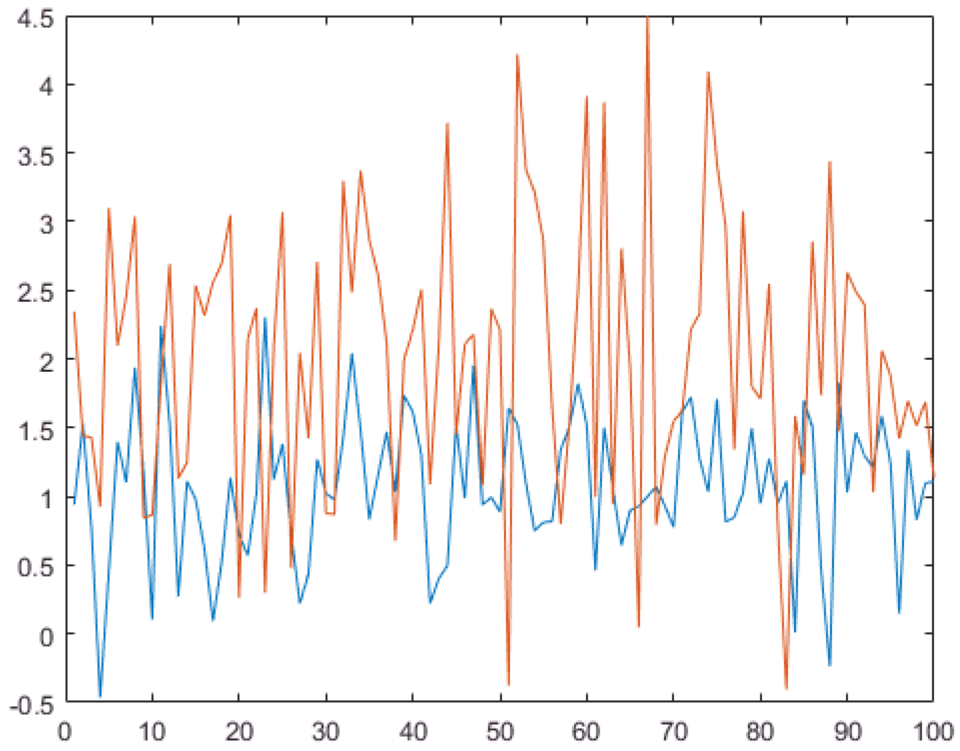
```
fitlm(BourkeN,BourkeS)
```

```
ans = 

Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE         tStat       pValue  
                   ________    _________    ______    __________

    (Intercept)    8.6732         2.2884      3.79    0.00015081
    x1             1.0535      0.0015121    696.72             0


Number of observations: 47950, Error degrees of freedom: 47948
Root Mean Squared Error: 366
R-squared: 0.91,  Adjusted R-Squared 0.91
F-statistic vs. constant model: 4.85e+05, p-value = 0
```

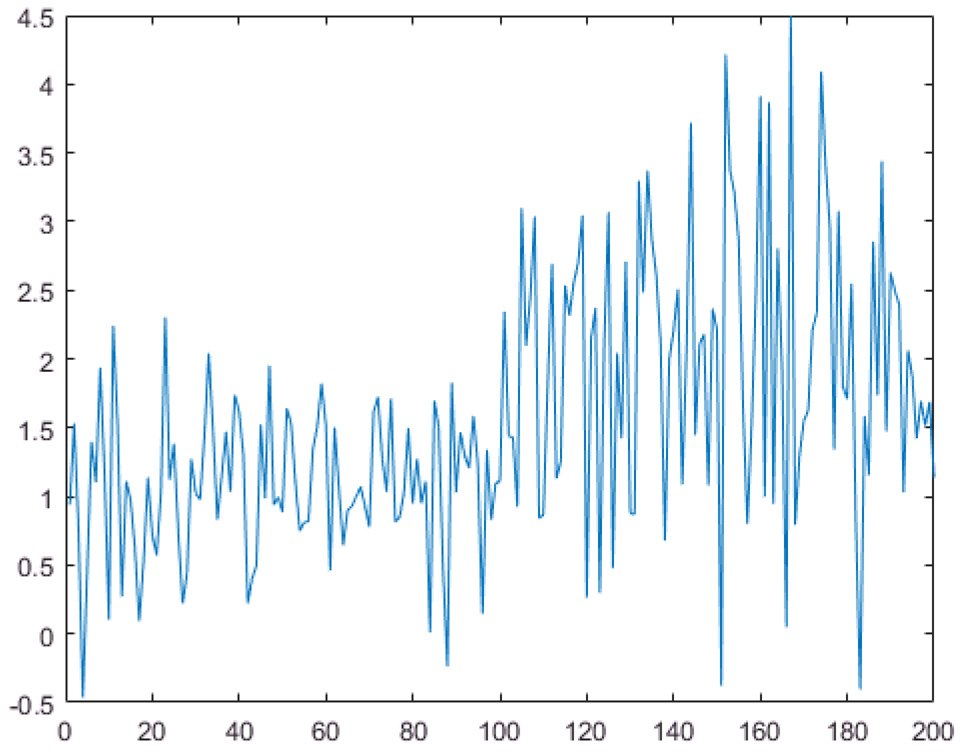
Logistic regression

We can also use `glmfit` for logistic regression, using the options 'binomial' and 'logit'. For an example, let's look at separating two Gaussian distributions.

```
% use randn to make two fake distributions
mu1=1;
sigma1=0.5;
mu2=2;
sigma2=1;
x1= mu1+sigma1.*randn(100,1);
x2=mu2+sigma2.*randn(100,1);
figure; plot(x1); hold on; plot(x2)
```

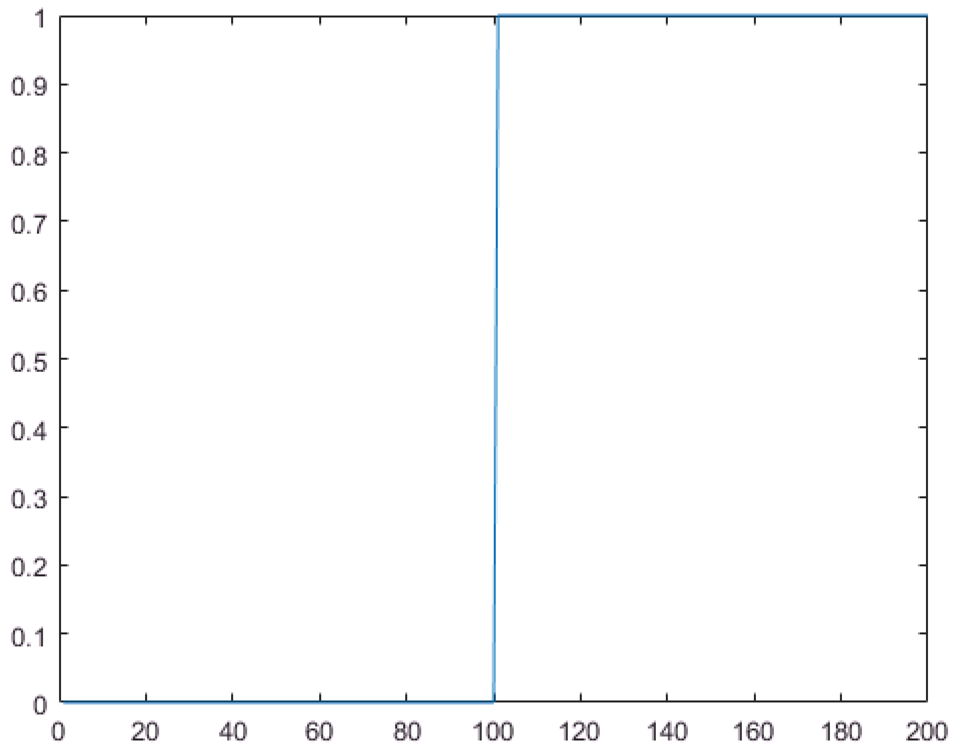


```
x= [x1;x2];
figure; plot(x)
```



Now we need to treat them as one variable, and create a model that predicts class 1 (x1) or class 2 (x2).

```
% this is our input

% this is our class label
% 0 = class 1
% 1 = class 2
class1 = zeros(size(x1));
class2 = ones(size(x1));
y= [class1;class2];
figure; plot(y)
```



```
% fit a logistic regresion model
logit_model= glmfit(x, [y ones(size(y))], 'binomial', 'link', 'logit')
```

```
logit_model =
   -2.3998
    1.6117
```

-

The model contains an intercept, and weights for the input (only one dimension in our case -) and is given by:

```

```

So to get the output of our model

```
% apply our weights
input= logit_model(1)+ logit_model(2).*x;
% run the model
output= 1./(1+exp(-input))
```

```
output =
    0.2922
    0.5171
    0.2377
    0.0412
    0.1594
    0.4633
    0.3496
    0.6725
    0.3755
    0.0968
      ⋮
```

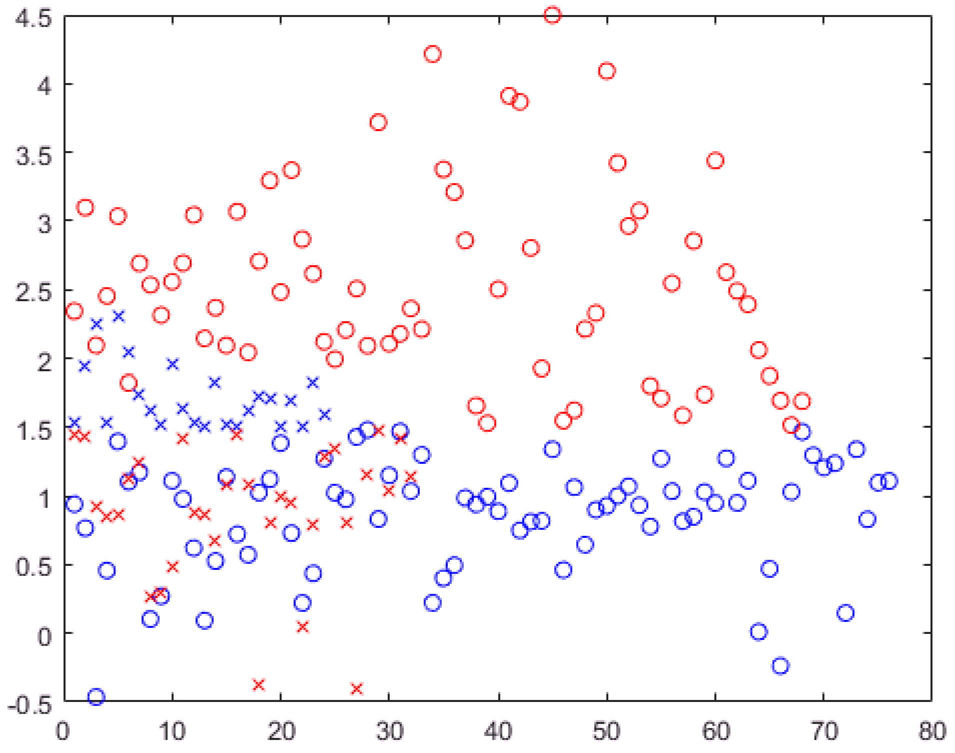
•

```

```

The output is between 0 and 1. It is usual to classify the input as class 1 (or Y = 0) for output < 0.5 and class 2 (or Y = 1) for output > 0.5.

Let's plot the results. We'll make class 2 red, and class 1 blue. We'll make the true predictions circles and the false ones crosses.

```
% these are correctly predicted values for class 2
% we'll make class red
plot(x(output<0.5& y==0), 'bo')
hold on
plot(x(output>=0.5& y==1), 'ro')
% these are the correctly predicted values for class 1

plot(x(output>=0.5 & y==0), 'bx')
plot(x(output<0.5 & y==1), 'rx')
```



```
% now the incorrect predictions for
% class 2 and class 1

```

We can also count how many we got right and wrong

```
% wrong
Nwrong= sum(output>=0.5 & y==0)+ sum(output<0.5 & y==1)
```

```
Nwrong = 56
```

```


Nright= sum(output<0.5&y==0)+sum(output>=0.5 & y==1)
```

```
Nright = 144
```

Challenge

```
%% CHALLENGE
% Using logistic regression to distinguish between
% two Gaussians with different means
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Use 80% of your data from Flinders street
% station between 8am - 9am to fit a logistic
% regression model


% Run your model on the remaining 20% of the data
% and classify it as either a weekend or weeday

%% EXTENSION
% how accurate is your classifier?

%% EXTENSION
% make your accuracy measure more robust using
% k-fold cross validation
% That means running your entire code (fit model,
% run model, test performance) in a loop.
% For each loop use a different partition of data
% to train and test. With 10 loops, train on
% 90% of the data, and test on 10%.
% With 5 loops, train on 80% of the data and
% test on 20%.

clear
close all
clc

load('PedCounts.mat')
 % Remove nans
nans = find(isnan(Sensor_ID));

Sensor_ID(nans) = [];
Hourly_Counts(nans) = [];
Date_Time(nans) = [];

% Get variables at flinders st
```

```
Dates_Flinders = Date_Time(Sensor_ID == 6);
Weekday_Flinders = weekday(Dates_Flinders);
Count_Flinders = Hourly_Counts(Sensor_ID==6);

% convert dates to date-vectors
Dates_Flinders = datevec(Dates_Flinders);
Hour_Flinders = Dates_Flinders(:,4);

% count at 8 - 9am
Flinders8 = Count_Flinders(Hour_Flinders == 8);
% count 11 - 12pm
Flinders11 = Count_Flinders(Hour_Flinders == 11);

% weekdays at FLinders at 8 in the morning
FlindersWeekday = Count_Flinders(Hour_Flinders == 8 ...
    & ismember(Weekday_Flinders,2:6));

FlindersWeekend = Count_Flinders(Hour_Flinders == 8 ...
    & ismember(Weekday_Flinders,[1,7]));

% Make vectors that will be used to train the classifier
X = [FlindersWeekday ; FlindersWeekend];
Y = [zeros(size(FlindersWeekday)) ; ...
    ones(size(FlindersWeekend))];

% Randomly permute the values to remove any patterns for the classifier
rand_ind = randperm(length(X));
X = X(rand_ind);
Y = Y(rand_ind);

Ntest = floor(0.2 * length(X)); % define the number of values we will need to test the classif

% In this part, we are going to split the data 5 times. Each time, 80% of the data will be use
% be used to test the material. This means that every data point will be used for training the

for partitions = 1:5

    % create
    test_ind0 = Ntest*(partitions-1) + 1;
    test_ind1 = Ntest*partitions;

    % set aside test data and labels
    test_data = X(test_ind0:test_ind1);
    test_labels = Y(test_ind0:test_ind1);
    % initialzie training data
    train_data = X;
    train_labels = Y;
    % remove the test cases
    train_data(test_ind0:test_ind1) = [];
    train_labels(test_ind0:test_ind1) = [];
    % run the logistic model
    log_model = glmfit(train_data,...
        [train_labels ones(size(train_labels))], ...
    'binomial','link','logit');


    % test our model
    input = log_model(1) + log_model(2)*test_data;
    output = 1 ./ (1 + exp(-input));
    % number correct
    Nright = sum(output > 0.5 & test_labels == 1) + ...
```

```
        sum(output <= 0.5 & test_labels == 0);
    % percentage correx
    100 * Nright / length(test_labels)

end
```

```
ans = 96.9574
ans = 95.5375
ans = 96.7546
ans = 96.7546
ans = 96.9574
```

Testing Gaussians

Many standard tests are built into MATLAB. For example, to test if data are normally distributed we can use `kstest`.

Another common test is `ttest2` (two-sampled t-test) and its non-parametric cousin `ranksum` (Wilcoxon rank sum / Mann-Whitney U test - why do statisticians like to name everything after themselves?).