COM2004/3004 Lab Week 2
Writing MATLAB programs – Understanding Sampling

Objectives

- **To provide an introduction to MATLAB programming.**
- **To gain an intuitive understanding of sample statistics.**

1. Background

In this lab class you'll be using data that represents the weights and lengths of male and female **Bengal tigers.** In order to build classifiers we are going to want to be able to estimate the means (i.e. averages), variances and covariance of these measurements. In order to discover the true means and variances we'd have to catch and measure every one of the roughly 2,500 tigers that are thought to exist – a project that is clearly impractical. So, instead we will have to be satisfied with an **estimate** of the true mean and variance that we generate by looking at the statistics of a small **sample** of tigers (e.g. we might catch and measure 20 animals). This is generally always going to be the case: we will be interested in the statistics of a large population, but we will have to make do with estimates derived from a smaller – often much smaller – sample.

In Part 1 of this lab class we will be using MATLAB to find out what happens when we try to compute estimates of statistics using small samples. We will see how the reliability of the estimates improves as the sample size increases. We will also see how estimates of variance and covariance can be biased (i.e. systematically too small or too large) if we don't handle the sample correctly.

Part 2 consists of some little MATLAB programming challenges. Spend no more than 90 minutes on Part 1 and at least 20 minutes on Part 2.

PART 1 – Sample Statistics

2. Loading and understanding the lab data

The tiger population data we are using has been `made up' (i.e. no-one really knows the weights and lengths of every living Bengal tiger) but it has been generated to be consistent with the ranges of weight and length that are quoted on Wikipedia. It has also been engineered so that the population statistics are nice round numbers, i.e. just to make things a little clearer.

The data can be downloaded from my web page. Visit,

```
http://staffwww.dcs.shef.ac.uk/people/J.Barker/COM2004.html
```

and right click to download and save the data `male_tigers.txt` and `female_tigers.txt` somewhere on the Windows file system (e.g. either on the Desktop or in your student filestore).

Navigate to the directory where you saved the data and load it using,

```
female_tigers = load('female_tigers.txt');
male_tiger = load('male_tigers.txt');
```

You will now have two matrices each with 1,250 columns and two rows. The columns represent the tigers (1,250 of each gender). The first row gives the weight (in kg) and the second the length (in cm). **`Tip`**: when displaying the data matrix it might be handy to transpose it first so that it has 1250 rows rather than columns. Also the command `more on` will put MATLAB into paging mode

```
more on;
female_tigers'
```

We have access to the data *for all the tigers* so we can compute the **true mean and variance** of the weight and length features. For example, for the female tigers try

```
mean(female_tigers, 2)
var(female_tigers, 0, 2)
```

We are now going to imagine that we don't know this truth but need to estimate it from a small sample of tigers that have been caught, weighed and measured.

3. Sampling the data

Imagine that you have gone out and caught 20 female tigers and measured them. You'd now know the measurements of 20 tigers of the 1250 that exist. Let's simulate this experiment by taking the first 20 of the female tigers from our complete population data

```
sample = female_tigers(:, 1:20);
```

From now on we'll just be considering the **tiger weight** feature. Let's compute the mean and variance of this feature in our sample,

```
mean(sample(1,:))
var(sample(1,:))
```

Notice how these estimates are not the same as the true mean and variance. But how close to the real values are they likely to be?

4. Repeated sampling

If we repeated our experiment and caught 20 different tigers we would get a different set of measurement and hence a different estimate of the mean and variance. Using MATLAB we can simulate running this experiment 1000's of times and getting 1000's of different estimates of the mean and variance. We can then plot a histogram of these estimates and compare them to the true values (which we wouldn't know in reality but which we computed above by using all the data).

So we will now repeat the experiment. In order to catch 20 different tigers we will **shuffle the data** before taking the first 20. We can do this by generating a vector containing the numbers 1 to 1250 in random order using the command `randperm(1250)` and then using this to reorder the data matrix

```
shuffled_tigers = female_tigers(:, randperm(1250));
sample = shuffled_tigers(:,1:20)
```

(Can you think of a slightly more efficient way to do this step still using randperm ?)

(Note, the MATLAB statistics toolbox – which is installed in the lab – has more efficient ways of drawing samples, but the code above will work even without the toolbox and is compatible with Octave.)

Now, compute the sample mean and variance for the tiger weights again and see that the result has changed.

Place all of this into a script (i.e. the shuffling, the sampling and the mean and variance calculation). Call it something like `calc_sample_stats.m`

Run the script over and over again. Note how each time you get a different mean and variance estimate.

5. Writing a sampling *function*

In order to make our `calc_sample_stats` script more useful we want to wrap it up into a MATLAB *function*. Functions can be called from other scripts and they can have input-arguments and return-values.

The function's inputs should be the complete population data and the sample size; the output should be the mean and variance estimates, i.e. the first line of the function should look like this,

```
function [sample_mean, sample_var] = calc_sample_states(data, N)
```

Write the complete function. Refer to the notes or ask a demonstrator if you are stuck.

Test the function

```
[sample_mean, sample_var] = calc_sample_stats(female_tigers, 20);
sample_mean
sample_var
```

Now open another script file – call it something like `do_sampling`. This script is going to call your `calc_sample_stats` function lots of times by using a loop. It will save the results of each run in a vector.

```
for i=1:10000
[sample_mean(i),sample_var(i)=calc_sample_stats(female_tigers, 20)
end
```

We can now plot a histogram of the results

```
hist(sample_mean, 100);
```

How does the distribution of the sample means compare with the true mean value?

```
hist(sample_var, 100);
```

How does the distribution of the sample variance compare with the true variance?

Calculate the average of the mean estimates, i.e. `mean(sample_mean)`. Are the estimates (on average) the same as the true mean value? Do the same for the variance estimates. Are the variance estimates on average the same as the true variance?

6. Notes on bias

You should have noticed in the previous section that the variance estimates were on average lower than the true variance. The estimate is *biased*. You can correct for this bias by multiplying by N/N-1 or, alternatively, by dividing by N-1 instead of N when you do the original variance calculation. MATLAB's var() command actually defaults to dividing by N-1 and hence computes an unbiased estimate. So to illustrate the bias issue in this lab class we have had to set the 2nd parameter to 1 to make it divide by N.

(Why does the sample variance under-estimate the true variance? Informal explanation: The point to remember is that the variance is the average squared distance of points from the true mean. When we take our sample we form an estimate of the mean. So when we estimate variance we are computing the distance of points to the *estimated* mean not the true mean. The estimated mean is going to be at the centre of our sample and will always be closer to the sample points than the true mean. This is seen clearly by taking the extreme case of a sample size of 1. For a single point the sample mean will always be the sample itself, i.e. the distance to the sample mean is 0 so the sample variance is 0 --- which is clearly a severe underestimate of the true variance.)

7. Experimenting with the sample size

Now let's experiment with different sample sizes and see how changing the sample size changes the distribution of the estimates of mean and variance.

We will use sample sizes of 2, 5, 10, 20, 40 and 100.

Modify your `do_sampling` code by wrapping the sampling loop in an outer loop that sets the sample size. We will want to be able to compare the histograms that are generated for each sample size, so use the subplot command to send each histogram to a different subplot in the same figure (i.e. same window). Note, you can open a new window for plotting by using the command `figure`. You can assign the window a number by using the form, `figure(1);figure(2)` etc. You can then direct plots to an existing window by reissuing the figure command, e.g. after `figure(1)` plots will be sent to window 1. Using `figure` and `subplot` see if you can arrange to have histograms for the sample means in one figure and histograms for sample variances in another.

What do you notice about how the distribution of the estimates changes as the sample size increases?

8. Classifying

Compare the histogram of weights of the female and male tigers for the entire population. A simple classifier could choose a threshold weight W and then if a tiger weighs more than W it would label it as male and if it weighs less than W it would label it as female. Looking at the histograms, what would be the best choice for the weight W?

Now take a sample of 100 female tigers and 100 male tigers. Plot histograms using these small samples. Looking at the histograms choose a threshold weight again. Is the threshold chosen using the small sample the same as the threshold you chose when using knowledge of the full set of data?

As we will see later in the module, having insufficient `training data' can lead to poor choices when designing classifiers.

PART 2 – Programming Challenge

9. Programming exercises

For the final stage of the lab class I have set four little programming exercises (see end). There are many ways of coding solutions to each of these exercises. Some solutions will be a lot more efficient than others. Learning how to write efficient code is one of the challenges of MATLAB.

For each program I'd like you to measure the runtime of your code. You can do this easily by using the MATLAB commands `tic` and `toc`. The command `tic` starts a timer and `toc` reports the time elapsed since the timer started, so you can roughly measure execution time using the following,

```
tic
somecode
toc
```

As an example we will consider the task of generating and storing 10,000,000 random numbers. I have shown three example solutions below. The solutions are functionally equivalent but have vastly different runtimes. Recent versions of MATLAB are better at optimizing out inefficiencies, but writing efficient code can still make a huge difference. For Octave this is even more true.

Method 1

```
% Using a loop - no matrix preallocation
tic
for n=1:10000000
x(n)=rand;
end
toc
```

Method 2

```
% Using a loop but preallocating storage
```

```
tic
x=zeros(1,10000000);
for n=1:10000000
x(n)=rand;
end
toc
```

Method 3

```
% Correctly using a built-in function to avoid the loop
tic
x=rand(1,10000000);
toc
```

The table shows execution times in seconds on an iMac using either MATLAB or Octave.

	Method 1	Method 2	Method 3
MATLAB	4.67	0.49	0.16
Octave	??? Gave up	130	0.26

Write solutions to the problems below in any way you please. I would like you to measure the execution times of your code for each exercise. If you finish before Friday then please email me your solutions and the runtimes. We will discuss solutions and how to write efficient code on Friday.

i/ Consider all the multiplications AxB for integers A and B lying in the range 1 to 1000. How many of these end in the digit 7. (Hint: You may want to experiment with the mod() function.)

ii/ Using the rand function generate 1,000,000 random numbers in the range 0 to 1 and then find the smallest and largest difference between a pair of adjacent numbers in the list.

iii/ Find the integers A and B in the range 1 to 1000 for which A/B lies closest to pi/4.

iv/ A random number is selected between 0 and 1 (with a uniform distribution). If a sequence of 4 such numbers are independently generated what is the probability that the sequence will be in ascending order? Estimate the probability by performing the experiment 1,000,000 times.

10. What next – Friday Lecture

MATLAB: Tips for writing efficient MATLAB code. Vectorizing your algorithms

Classification: Final `review' lecture: elements of probability theory.