Univariate Gaussians

The Gaussian distribution, also called the normal distribution, is widely used in probabilistic machine learning. This week we'll see that you should know about Gaussians when doing some basic statistics on your experimental results. Later in the course we'll use Gaussians as building blocks of probabilistic models, and to represent beliefs about unknown quantities in inference algorithms.

As large parts of the MLPR course depend on thoroughly understanding Gaussians, I have written quite detailed (slow) notes compared to the machine learning text books, and provided exercises. A later note on multivariate Gaussians will also be important. Many of you will have seen all of this material before (some of you several times), so I won't step through it all in lectures. However, I am happy to answer questions on the class forum.

The standard normal distribution, the zero-mean unit-variance Gaussian

I think of a probability distribution as an object, like a black box, that outputs numbers. A number-producing box for the standard normal distribution has a label, $\mathcal{N}(0,1)$, and a button. If we press the button, the box displays a number. You can simulate this process at a Matlab/Octave prompt by typing `randn()`. You can also use `randn()` in Python after typing: `from numpy.random import randn`. If you 'press the button' multiple times, or run `randn()` multiple times, you'll get different numbers. To attach a label to an outcome from the generator we write in mathematics $x \sim \mathcal{N}(0,1)$, or in code `x = randn()`.

Each call to `randn()` gives a different number, and many calls reveal the distribution encoded in the generator, described by the label $\mathcal{N}(0,1)$. A histogram of a million draws from the distribution, reveals a classic 'bell-curve' shape. You should be able to sketch this bell-shaped curve, with points of inflection at ± 1. Try plotting a histogram of a million standard normal samples on a computer now. If you know how to do it, opening a terminal and creating the plot can be done in about 10 seconds. If you don't know how to do it, learning to sample test data and to create plots is an important skill to develop. Example code is at the end of this document.

The observed or empirical mean of the samples will be about 0, and the empirical variance will be about 1. (Check this on your samples!) In the limit of infinitely many samples these values become exact, hence the numbers in the label $\mathcal{N}(0,1)$. Formally: $\mu = E[x] = \int x\, p(x)\, \mathrm{d}x = 0$ and $\mathrm{var}[x] = E[(x-\mu)^2] = \int x^2\, p(x)\, \mathrm{d}x = 1$. There is a separate sheet on working with expectations if you need more review of probability.

The 'probability density function' (PDF) for the standard normal distribution describes the bell-shaped curve:

$$p(x) = \mathcal{N}(x; 0, 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

The probability of landing in a narrow range of width δ between $x - \delta/2$ and $x + \delta/2$ is about $p(x)\delta$. This statement becomes accurate in the limit $\delta \to 0$. So with $N = 10^6$ samples, we expect about $p(x)N\delta$ samples to land in a narrow histogram bin of width δ at position x. Using this approximation, you should be able to plot a theoretical prediction through the histogram you produced earlier.

Shifting and scaling: general univariate Gaussians

[The separate notes on expectations contain more material on shifting and scaling random variables.]

We can define a process to generate a quantity y that draws a value from a standard normal, $x \sim \mathcal{N}(0,1)$, and then adds a constant $y = x + \mu$. The mean of this distribution, is μ.

If we drew many values from $\mathcal{N}(0,1)$, and added some constant μ to each one, the histogram of the values keeps the same shape and simply shifts to be centred at μ. The PDF will shift in the same way. This distribution has the same shape, just with a different location, and so is still called a Gaussian, just with mean μ instead of mean 0. To get the PDF of the new variable, we replace every occurrence of x with $(y-\mu)$:

$$p(y) = \mathcal{N}(y;\mu,1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-\mu)^2}.$$

The choice of variable names, such as x or y, is arbitrary in this note. You will of course also see x used for Gaussians with non-zero mean.

Similarly, we can define a random variable z that multiplies or 'scales' a standard normal outcome by a constant σ, before adding a constant μ:

$$z = \sigma x + \mu.$$

If we scale and shift many draws from a standard normal, the histogram of values will be stretched horizontally, and then shifted. Scaling by σ multiplies the variance by σ^2 (see the notes on expectations), and leaves the mean at zero. Adding μ doesn't change the width of the distribution, or its variance, but adds μ to the mean.

The distribution of z maintains the same bell-curve shape, with the points of inflection now at $\pm\sigma$ (note, *not* $\pm\sigma^2$). We still say the variable is Gaussian distributed, but with different parameters: $z \sim \mathcal{N}(\mu, \sigma^2)$. By convention, the second parameter of the normal distribution is usually its variance σ^2, not its width or standard deviation σ. However, if you are reading a paper, or using a new library routine, it is worth checking the parameterization being used, just in case. Sometimes people choose to define a Gaussian by its precision, $1/\sigma^2$, instead of the variance.

For a general univariate Gaussian variable $z \sim \mathcal{N}(\mu, \sigma^2)$, we can identify a standard normal, by undoing the shift and scale above:

$$x = \frac{z-\mu}{\sigma}.$$

Substituting this expression into the PDF for the standard normal, suggests

$$p(z) = \mathcal{N}(z;\mu,\sigma^2) \propto e^{-\frac{1}{2\sigma^2}(z-\mu)^2}.$$

However, we have to be careful transforming random variables. Here, stretching out the bell-curve to be σ times wider would make the area underneath it σ times bigger. However, PDFs must be normalized: the area under the curve must be one. To conserve probability mass, if we stretch a region of outcomes, we must also decrease the PDF there by the same factor. Hence, the PDF of a general (univariate) Gaussian is that for a standard normal, scaled down by a factor of σ:

$$p(z) = \mathcal{N}(z;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(z-\mu)^2},$$

which can also be written by replacing the σ in the denominator with σ^2 inside the square-root.

Check your understanding further

- What's the difference between $\mathcal{N}(0,1)$ and $\mathcal{N}(x;0,1)$?
- What is the integral $Z = \int e^{-\frac{1}{2\sigma^2}(u-\mu)^2} \, \mathrm{d}u$? By considering $\int \mathcal{N}(u;\mu,\sigma^2) \, \mathrm{d}u$, you should be able to write down the answer without doing any detailed calculations.
- What is the integral $C = \int \sum_{k=1}^{K} w_k \mathcal{N}(x;\mu_k,\sigma_k^2) \, \mathrm{d}x$, for a general set of weights w_k? Again, no detailed calculations should be required.

Further reading

https://en.wikipedia.org/wiki/Normal_distribution

Matlab/Octave code

To generate a million outcomes from a standard normal and plot a histogram:

```
N = 1e6;
xx = randn(N,1);
hist(xx, 100); % 100 bars shows shape better than default of 10
```

To check the mean and variance:

```
empirical_mean = mean(xx)
empirical_var = var(xx)
```

You should understand how to plot a theoretical prediction of this histogram shape. Fill in the parts marked TODO below:

```
[cc, bin_centres] = hist(xx, 100);
% Fill in an expression to evaluate the PDF at the bin_centres.
% To square every element of an array, use .^2
pdf = TODO;
bin_width = bin_centres(2) - bin_centres(1);
predicted_bin_heights = TODO; % pdf needs scaling correctly
% Finally, plot the theoretical prediction over the histogram:
hold on;
plot(bin_centres, predicted_bin_heights, '-r');
hold off;
```

Python code

To generate a million outcomes from a standard normal and plot a histogram:

```
import numpy as np
from matplotlib import pyplot as plt

N = int(1e6) # 1e6 is a float, numpy wants int arguments
xx = np.random.randn(N)
hist_stuff = plt.hist(xx, bins=100)
plt.show()
```

To check the mean and variance:

```
print('empirical_mean = %g' % np.mean(xx)) # or xx.mean()
print('empirical_var = %g' % np.var(xx))   # or xx.var()
```

You should understand how to plot a theoretical prediction of this histogram shape. Fill in the parts marked TODO below:

```
bin_centres = 0.5*(hist_stuff[1][1:] + hist_stuff[1][:-1])
# Fill in an expression to evaluate the PDF at the bin_centres.
# To square every element of an array, use **2
pdf = TODO
bin_width = bin_centres[1] - bin_centres[0]
predicted_bin_heights = TODO # pdf needs scaling correctly
# Finally, plot the theoretical prediction over the histogram:
plt.hold('on')
plt.plot(bin_centres, predicted_bin_heights, '-r')
plt.hold('off')
plot.show()
```