Machine Learning and Pattern Recognition
Tutorial 5

Instructor: Iain Murray

Motivation for this tutorial: Some of the questions in this tutorial ask you to perform some (short!) numerical experiments. Hopefully learning to do these things quickly will be a useful skill. Putting standard error bars on experimental results is a minimal piece of statistical care which might be useful in many of your projects.

1. **Model comparison computation:** It's common to compute log-likelihoods and log-marginal-likelihoods to avoid numerical underflow. (Quick example: notice that the probability of 2000 coin tosses, 2^{-2000}, underflows to zero in Matlab, or any package using IEEE floating point.)

   Assuming there are only two possible models, M_1 and M_2, we define:

   $$a_1 = \log P(D \mid M_1) + \log P(M_1)$$
   $$a_2 = \log P(D \mid M_2) + \log P(M_2).$$

   These 'activations' are the log-posteriors of each model, up to a constant. Show that we can get the posterior probability of model M_1 neatly with the logistic function:

   $$P(M_1 \mid D) = \sigma(a_1 - a_2) = \frac{1}{1 + \exp(-(a_1 - a_2))}.$$

   Given K models, with $a_k = \log[P(D \mid M_k)\, P(M_k)]$, show:

   $$\log P(M_k \mid D) = a_k - \log \sum_k \exp a_k.$$

   The $\log \sum \exp$ function occurs frequently in the maths for probabilistic models (not just model comparison). Show that:

   $$\log \sum_k \exp a_k = \max_k a_k + \log \sum_k \exp \left(a_k - \max_{k'} a_{k'} \right).$$

   Explain why the expression is often implemented this way. (Hint: consider what happens when all the a_k's are less than -1000).

2. **Simple Monte Carlo:** Let x be distributed according to a distribution with mean μ and variance σ^2. To construct an example, sample a vector of a hundred samples from Uniform[0,1], using `rand` in Octave or Matlab. Given access to only these samples, find estimates $(\hat{\mu}, \hat{\sigma}^2)$ of the mean and variance of x.

   Show that the Monte Carlo estimate of the mean

   $$\mathbb{E}[x] \approx \frac{1}{S} \sum_{s=1}^{S} x^{(s)}$$

   has variance σ^2 / S (in general, not just for a uniform).

   It's common to report an estimate of a mean using a 'standard error', the square root of an estimate of this variance:

   $$\mathbb{E}[x] = \hat{\mu} \pm \frac{\hat{\sigma}}{\sqrt{S}}$$

   Write down your estimate of the mean in this form. Is your answer within two standard errors of the true answer? It should fall in that range around 95% of the time. In a class

of 120, some of you will probably get a 'wrong' estimate! Try running your code again and see if the true answer is *usually* within 1 or 2 standard errors.

Also report an estimate of $\mathbb{E}[x^2]$ with a standard error.

3. **Rejection sampling:** use a simple Monte Carlo procedure to estimate the mean of the truncated normal:

$$p(x) \propto \begin{cases} N(x; 0, 1) & x > 1 \\ 0 & \text{otherwise} \end{cases}$$

What would happen if you applied your procedure to a truncation at $x > 6$?

4. **Importance sampling:** How could we use importance sampling to estimate the mean of a truncated distribution with $x > 6$?

5. **Metropolis sampling:** we could apply the Metropolis algorithm (using the code from the slides and copied below, or otherwise) to estimate the mean of the truncated normal with $x > 6$.

What would be the difficulty with reporting error bars on our estimate?

```
function samples = dumb_metropolis(init, log_ptilde, iters, sigma)

D = numel(init);
samples = zeros(D, iters);

state = init;
Lp_state = log_ptilde(state);
for ss = 1:iters
    % Propose
    prop = state + sigma*randn(size(state));
    Lp_prop = log_ptilde(prop);
    if log(rand) < (Lp_prop - Lp_state)
        % Accept
        state = prop;
        Lp_state = Lp_prop;
    end
    samples(:, ss) = state(:);
end
```