Markov Chain Monte Carlo Sampling Algorithms & Machine Learning Applications

Student Number 201503311

Michael G. Turner

A thesis presented for the degree of
Masters in Mathematics

School of Mathematical & Physical Sciences
University of Hull
United Kingdom
3rd of May 2019

Markov Chain Monte Carlo Sampling Algorithms & Machine Learning Applications

Student Number 201503311

Michael G. Turner

Abstract

This report will go through the mathematics that are used in different computational random sampling methods known as Markov Chain Monte Carlo sampling algorithms or MCMC. I will present how these algorithms are implemented in the programming language R with different examples that will visualise how these algorithms work. I will discuss the work I have done into how machine learning classification works and go through some examples of different applications where they are used through another programming language Python. Now, without further ado, we shall begin.

Contents

Chapter 1

Introduction

Before diving into the details of computational random sampling methods I would like to go over some of the basic statistics you will need to understand what will be covered throughout this entire report.

1.1 The Frequentist Vs The Bayesian

The very first thing to clarify is how Bayesian is different from Frequentist. The Frequentist views probability as being the *relative frequency* of an event occurring in an infinitively long series of *identical* trials. This can be represented in a mathematical expression as

$$P(h) = \lim_{n\to\infty} \frac{h}{n}$$

where h is the number of times that the event occurred and n is the total number of trials. This might appear perfectly logical but the trouble lies in the assumptions, one of which says that the trials are *identical*. This is bad news because, in the real world, the different variables will not be identical each time we flip the coin.

Suppose that I flipped a coin at 67.3cm from the ground and the coin had an orientation to the ground that was an angle of 32° relative to the horizontal and 2° relative to the vertical. If I repeated this set up *exactly* then I should expect to get a certain value out with every flip because the system is in itself deterministic, right? This is where we encounter issues with the Frequentist view with what is truly meant by the term *identical*.

We could possibly determine the coin to be a set height from the ground but allow the orientation to alter, but then we start to enter a subjective view for what identical means which goes against the Frequentist belief. For a real life scenario like this there is always going to be a sense of uncertainty because the experiment cannot be recreated exactly as before. With a Bayesian perspective, we can avoid this issue.

For Bayesian, we think of the probability of an event occurring as the number of events divided by the number of possible outcomes, which are all to be *equally* likely. What this

means is that we could quantify all the possible orientations of the coin, which will be the initial conditions, and then look at the forces acting on the coin. We can combine these and then think about, for each of these conditions, what value will the coin give? So the number of possibilities is given by the number of initial conditions and the number of heads represents the total number of heads we get across all these different possibilities.

Like with the Frequentist approach, there are some assumptions being made here as well which is that the data is fixed. What that would mean is that, given some initial condition like the orientation and height of the coin, we would get the same value every time. But the reason we do not is because the parameters can vary.

This is much more realistic approach as the Bayesian probability does not represent a long run frequency of getting a head but rather an uncertainty. We do not know the exact initial orientation of the coin so we are left with some underlying uncertainty with whether we will get a head or a tails.

It could be regarded as common sense to know that you can never be *completely* certain of an event happening. And it is also important that the Bayesian view does not rely on an infinite number of samples. Bayesian probability relies heavily on something called conditional probability.

1.2 Conditional Probability

Conditional probability of, say an event B, is the probability that an event B will occur conditional on another event, say A. The event B will *depend* on event A. This is written as $P(B \mid A)$, and said as "the probability of B given A". If the two events are **independent** of each other then the probability of event B occurring given event A is simply just the probability of event B occurring, written as $P(B)$. The formal definition of Bayes' Theorem is given below.

Theorem 1.2.1. ***Bayes' Theorem:*** *Given a parameter θ and some data, the relationship between the probability of θ before the data, $P(\theta)$, and the probability of θ given the data, $P(A \mid data)$, is given by*

$$\overbrace{P(\theta \mid \text{data})}^{\text{Posterior}} = \frac{\overbrace{P(\text{data} \mid \theta)}^{\text{Likelihood}} \cdot \overbrace{P(\theta)}^{\text{Prior}}}{\underbrace{P(\text{data})}_{\text{Evidence}}} \tag{1.1}$$

(See [1] for a review)

Now there is a slight technicality here, we have to bring in the *prior* to perform the analysis. This represents the *assumption* for the value of θ before viewing the data. The prior is therefore frowned upon by the frequentist because there is the arguable idea that it brings in subjectivity into the strict manner of probability. But, then again, a prior can be just another assumption made while modelling the situation. Which would then make it just as objective as any of the other assumptions made in the frequentist model.

So just think of a prior as an assumption related to what we assume the probability to be distributed as. For example, if it was rainy today then I can assume the probability that it will be sunny tomorrow increases because the days are not independent. It has to stop raining at some point because the clouds will not reset over night to have the same amount of water. The resource is not unlimited just like everything in nature.

1.2.1 The Prior, Likelihood and Posterior

These terms were being brought up in the previous writing so, to be sure everyone reading knows what they mean, I will go through them now.

- The **prior** is what reflects the information for some parameter before viewing the data. If nothing is known about the parameter then we use a **flat prior** which does not expose much information.
- The **likelihood** is the way of introducing the data to the process. It represents how *plausible* the observed data is given a set of parameters.
- The **posterior** is a probability distribution for the parameters in the model given the data. It is what we know about the problem and relates to the prior and the likelihood. If the prior and likelihood are both vague then we will get a posterior that reflects these vague beliefs. It can be thought of as the updated version of the prior.

The **evidence** is the probability of getting the data averaged over all possible parameter values. It can be thought of simply as a **normalising factor**. It makes the integral of the posterior equal to 1. (If you are thinking "wait, what?" I will explain in the next section.)

(See [2] for more detail of §1.2)

The evidence term in Bayes' Theorem can lead to problems that, when we apply standard Bayesian methods, make the posterior term too difficult to calculate exactly by hand in equation (1.1). These more complex problems are approached using numerical MCMC sampling algorithms, which is what this report is all about!

1.3 Tricky Denominators

For a continuous parameter problem, Bayes' Theorem can become much more difficult to calculate since we will need to integrate. This is due to the denominator of Bayes' Theorem (equation 1.1) being related to the other probabilities as

$$P(data) = \int_{\text{All } \theta'} P(data \mid \theta') \cdot P(\theta') \, \mathrm{d}\theta' \tag{1.2}$$

Just observing the general form starts to indicate the difficulty we could face later. There are forms where the prior relates to the likelihood known as a **conjugate prior**. This results in the posterior being of the same form as the prior, effectively removing any complicated integral expression and allowing for a more simple exact analytical solution.

We are going to focus the remainder of this report on cases where we don't have a conjugate prior. The following example, which might appear simple enough initially, can become a problem that is difficult to solve. Consider the following equation:

$$P(\text{data}) = \int_{-\infty}^{\infty} \frac{(\theta')^6}{\sqrt{2\pi}} \cdot \exp\left(-\frac{(\theta')^2}{2}\right) \mathrm{d}\theta' \tag{1.3}$$

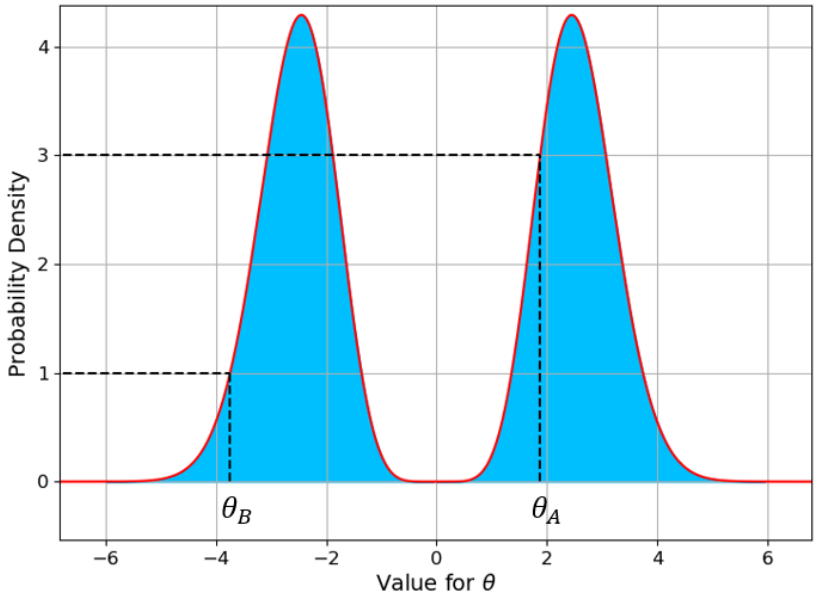


Figure 1.1: Plot of the integrand in equation (1.3) shown by the red line with the integral being the blue shaded area under graph. It also shows what two values in parameter space would give the frequency 3 and 1.

The reason we divide by this integral is to make the blue area under the graph equal to 1. This is what the term normalise refers to. Currently, the area is not equal to one as this is just the plot of what the numerator in Bayes' Theorem would be for this example

$$P(\text{data} \mid \theta) \cdot P(\theta) = \frac{\theta^6}{\sqrt{2\pi}} \cdot \exp\left(-\frac{\theta^2}{2}\right)$$

The integral in equation (1.3) is difficult to solve analytically so we need another way. Consider 2 points in parameter space and their ratio:

$$\frac{P(\theta_A \mid \text{data})}{P(\theta_B \mid \text{data})} = \frac{3}{1}$$

So the sampler would need to generate random samples three times as often from point θ_A than point θ_B. This is visualised in Figure 1.1. Now we want to use this in practice. But how can we use the relative posterior density to sample from the posterior?

Firstly, let us think more about what the posterior distribution actually is. It is the frequency that we will sample a value for an infinite sample of observations. We can use the relative density of a point versus all others to determine its sampling frequency. To

determine the frequency of samples at a given point in parameter space, we do not need the absolute value of the posterior at that point, if we know its value relative to other parameter values.

So, even though we cannot calculate the absolute posterior in general, we can still determine its value. Taking another two points in parameter space, θ_C and θ_D, we can calculate the ratio of the posterior density at these two parameter values using Bayes' Theorem (equation 1.1)

$$\begin{aligned}\frac{P(\theta_C \mid \text{data})}{P(\theta_D \mid \text{data})} &= \frac{\frac{P(\text{data} \mid \theta_C) \cdot P(\theta_C)}{P(\text{data})}}{\frac{P(\text{data} \mid \theta_D) \cdot P(\theta_D)}{P(\text{data})}} \\ &= \frac{P(\text{data} \mid \theta_C) \cdot P(\theta_C)}{P(\text{data} \mid \theta_D) \cdot P(\theta_D)} \end{aligned} \tag{1.4}$$

The tricky denominator term has cancelled out and now everything in equation (1.4) is known which gives us the un-normalised posterior. This is enough to give us the sampling frequency at each point in parameter space versus all other points. The reason that the un-normalised posterior tells us everything we need to know about the posterior is because all the shape of the posterior is determined by the numerator of Bayes' Theorem. So now we can write

$$\begin{aligned} P(\theta \mid \text{data}) &= \frac{P(\text{data} \mid \theta) \cdot P(\theta)}{P(\text{data})} \\ &\propto P(\text{data} \mid \theta) \cdot P(\theta) \end{aligned} \tag{1.5}$$

By only using the likelihood and prior, we can produce an un-normalised posterior which will have all the same properties like the mean and standard deviation. The **mean** is the average of the data and the **standard deviation** is the amount of variation in the data.

(See [3] Chapter 12 for more explanation on dependent sampling)

This chapter has covered all of the important tools that will be used throughout. A quick note that if you are unfamiliar with any of the probability distributions mentioned in this report there are many resources available online where you can learn about each one. Now we are ready to introduce some different sampling algorithms and how they are used in different applications.

1.4 Markov Chains

For completeness, I will define what a Markov chain is as the term state is used in definition 2.2.1 which relates to this definition. Do not worry if the mathematics is too advanced for you, it is not essential to understand in order to understand the rest of this report.

Markov Chain Monte Carlo (MCMC) is a process for generating samples $x^{(i)}$ while *wondering* the possible states $X = \{x_1, x_2, \ldots, x_n\}$ using the Markov chain mechanism which is constructed so that the samples $x^{(i)}$ replicate samples drawn from the target

distribution $P(x)$. A Markov chain on a finite state space where $x^{(i)}$ can take any state X is called a Markov chain if

$$P(x^{(i)} \mid x^{(i-1)}, x^{(i-2)}, \ldots, x^{(1)}) = P(x^{(i)} \mid x^{(i-1)}) \tag{1.6}$$

The progression of the Markov chain in a state space X depends only on the current state of the chain and a fixed **transition** matrix, denoted $\mathbf{P}$. To gather a more intuitive understanding we should see an example.

1.4.1 What's for dinner?

Consider a Markov chain with 3 states ($n = 3$). The corresponding transition matrix, illustrated by Figure 1.2, represents the probabilities of what I am hungry for, given the food I had on the preceding day

$$\mathbf{P} = \begin{bmatrix} & P & B & H \\ P & 0.5 & 0.3 & 0.2 \\ B & 0.25 & 0.5 & 0.25 \\ H & 0.4 & 0.4 & 0.2 \end{bmatrix}$$

This defines the probability of going from a state x_i to x_j where $i, j = 1, 2, \ldots, n$. So the element $[\mathbf{P}]_{ij}$ is the probability that a day with dinner i will be followed by a day with dinner j e.g. $[\mathbf{P}]_{12}$ is a 30% chance that a pizza will be followed by a burger.

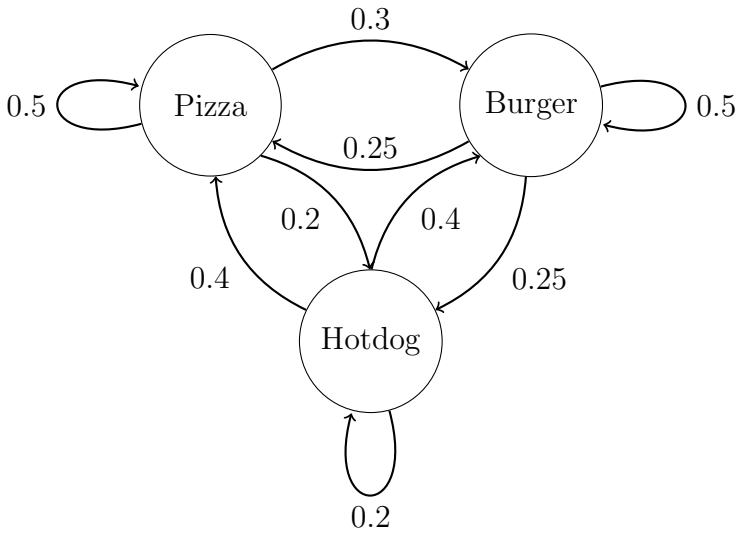


Figure 1.2: Graph for the transition of state with their corresponding probability.

So, say I have pizza for dinner on day 1. This is represented as the vector

$$x^{(1)} = [1, 0, 0]$$

Now the dinner I will have on day 2 is given by

$$x^{(2)} = x^{(1)}\mathbf{P} = [1, 0, 0] \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.25 & 0.5 & 0.25 \\ 0.4 & 0.4 & 0.2 \end{bmatrix} = [0.5, 0.3, 0.2]$$

There is a 50% chance I will have pizza again on day 2.

The dinner for day 3 is similar

$$x^{(3)} = x^{(2)}\mathbf{P} = x^{(1)}\mathbf{P}^2 = [1, 0, 0] \begin{bmatrix} 0.405 & 0.38 & 0.215 \\ 0.35 & 0.425 & 0.245 \\ 0.38 & 0.4 & 0.22 \end{bmatrix} = [0.405, 0.38, 0.215]$$

So the general formula for day n is

$$x^{(n)} = x^{(1)}\mathbf{P}^n$$

As the days go on, the vector $x^{(n)}$ tends to what is called a **steady state** vector which represents the probability of what I will have for dinner on all days, with the assumption that this is all I will eat for the rest of my days, no matter what the initial meal (distribution) is. This is a very important piece of MCMC simulation. For any initial distribution, the Markov chain will converge to the invariant distribution $P(x)$, represented as

$$P(x) = P(x)\mathbf{P} \tag{1.7}$$

Thus,

$$\begin{aligned} P(x)\mathbf{P} &= P(x) = P(x)\mathbf{I} \\ \Rightarrow \quad P(x)(\mathbf{P} - \mathbf{I}) &= P(x) \left(\begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.25 & 0.5 & 0.25 \\ 0.4 & 0.4 & 0.2 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \\ &= P(x) \begin{bmatrix} -0.5 & 0.3 & 0.2 \\ 0.25 & -0.5 & 0.25 \\ 0.4 & 0.4 & -0.8 \end{bmatrix} = \mathbf{0} \end{aligned}$$

Note the condition $P_1 + P_2 + P_3 = 1$. Using this we can solve these simultaneous equations to give us the steady state distribution

$$[P_1, P_2, P_3] = [0.3774, 0.4025, 0.2201]$$

This means that, as the days go on, I will have pizza for 37.74%, burger for 40.25% and hotdog for 22.01% of days in the future. It is important to note that Equation (1.7) only holds if the transition matrix $\mathbf{P}$ satisfies the following conditions:

- Irreducibility: For any state of the Markov chain, there is a positive probability of visiting all other states. That is, the matrix T cannot be reduced to separate smaller matrices.
- Aperiodicity: The chain should not get trapped in cycles.

A sufficient, but not necessary, condition to ensure that a particular $P(x)$ is the desired invariant distribution is the following reversibility condition

$$P\left(x^{(i)}\right)\mathbf{P}\left(x^{(i-1)} \mid x^{(i)}\right) = P\left(x^{(i-1)}\right)\mathbf{P}\left(x^{(i)} \mid x^{(i-1)}\right) \tag{1.8}$$

Summing both sides over $x^{(i-1)}$ gives

$$P\left(x^{(i)}\right) = \sum_{x^{(i-1)}} P\left(x^{(i-1)}\right)\mathbf{P}\left(x^{(i)} \mid x^{(i-1)}\right)$$

(See [4] for more detail)

1.5 Coding

We are going to focus on how these algorithms are used for different problems. All of these methods are numerical approaches so this report will heavily involve numerical implementation which I did in **R** and **Python**. If you have never used (or even heard of) these programming languages that is okay. You will still understand everything from a mathematical perspective. If you are familiar with these languages then you will be able view all of the code that I have written myself through the following link:

https://github.com/iMikeT/MathProject2

To run the different Python codes that I have written on your own computer you will need the package Scikit-Learn. And for the R codes I have used the plotting package ggplot2. You will need to install these yourself before running any code. The links to access specific code files for the example I cover will be provided throughout this report.

I have been teaching myself for the last two years how to write in these programming languages, so that I would be able to fully explore the functionality of all the different samplers involved and to move further into the world of machine learning which will come towards the end of this report.

All of the figures presented are my own, generated within the code mentioned above, and links to different animations that I have made will also be available within my GitHub repository link.

I have been able to write all of the Python code for the machine learning processes used in this report through taking free online courses given by edX in collaboration with Microsoft, one of which that was most helpful I will leave a link for you to see here.

Chapter 2

Metropolis Algorithms

These algorithms are the most popular MCMC algorithms for their simplicity. But for some problems these samplers will struggle and we will need something more advanced.

2.1 Random Walk Metropolis

As discussed before, **independent sampling** is generally impossible, and therefore cannot determine exactly the entire structure of the posterior. But we can generate an estimation for it with local exploration using **dependent sampling** i.e. the next sample value will *depend* on the current value. This is easier to implement since it only requires calculation of the un-normalised posterior.

But how do we choose *where* to propose a next step from the current and *whether* we move to it or stay? We need to sample more from the peaks (accept new step) and less from the lower levels (reject or accept) of the posterior probability density function (PDF). This leads us to the definition of the Random Walk Metropolis algorithm.

Definition 2.1.1. ***Metropolis:*** *Let $f(\theta)$ be a function that is proportional to the posterior distribution $P(\theta \mid data)$ such that*

$$P(\theta \mid data) \propto f(x) = P(data \mid \theta) \cdot P(\theta)$$

where θ is an n-dimensional, continuous parameter, and $n \in \mathbb{N}$ where $\mathbb{N}$ denotes the natural numbers.

1. *Choose an arbitrary point θ_0 for the first sample, and choose a proposal distribution $Q(\theta' \mid \theta)$ that suggests a candidate for the next sample value θ', given the previous value θ. For this algorithm, Q must be symmetric which means that $Q(\theta' \mid \theta) = Q(\theta \mid \theta')$. A typical choice is the Normal distribution centred at θ.*
2. *For each iteration t where $t \in \mathbb{Z}$, $t \geq 0$:*
    - *Generate a candidate θ' for the next sample by sampling from the proposal distribution $Q(\theta' \mid \theta_t)$.*

- *Calculate the acceptance ratio* $r = \dfrac{f(\theta')}{f(\theta_t)}$ *where* r *is the probability that the point in parameter space* θ' *is accepted as the next sample value. The acceptance ratio will decide to either accept or reject the candidate. Since* $f(\theta)$ *is proportional to* $P(\theta \mid data)$*, we have* $r = \dfrac{P(\theta' \mid data)}{P(\theta_t \mid data)}$ *since any constant of proportionality would be cancelled.*
- *Accept or reject:*
  - *Generate a uniform random number* $u \in [0, 1]$.
  - *If* $u \leq r$ *accept the candidate:* $\theta_{t+1} = \theta'$.
  - *If* $u > r$ *reject the candidate:* $\theta_{t+1} = \theta_t$.

Here $\mathbb{Z}$ *denotes the integers.*

Essentially, we want to take a "random walk" through parameter space where the next step is dependent on the current step and the shape of the posterior. The above can be formalised as:

$$r = \begin{cases} 1, & \text{if } P(\theta_{t+1} \mid \text{data}) \geq P(\theta_t \mid \text{data}) \\ \dfrac{P(\theta_{t+1} \mid \text{data})}{P(\theta_t \mid \text{data})}, & \text{if } P(\theta_{t+1} \mid \text{data}) < P(\theta_t \mid \text{data}) \end{cases} \tag{2.1}$$

We can substitute in the un-normalised posterior densities because the denominator cancels out as shown in equation (1.4). To visualise this process more clearly, we can consider an example.

2.1.1 A Stroll Through a Strange Park

Suppose we have a posterior density function that is not normalised. This means that we have the following:

$$P(x \mid \text{data}) = P(\text{data} \mid x) \cdot P(x)$$

taking x as the parameter variable instead of θ. Figure 2.1 shows a plot of the following equation:

$$P(\text{data} \mid x) \cdot P(x) = \frac{\exp\left(-\dfrac{(x-0.2)^2}{0.002}\right)}{\sqrt{\dfrac{6}{5}\pi}} + \frac{\exp\left(-\dfrac{(x-0.45)^2}{0.02}\right)}{\sqrt{\dfrac{2}{5}\pi}} + \frac{\exp\left(-\dfrac{(x-0.7)^2}{0.002}\right)}{\sqrt{\dfrac{4}{5}\pi}}$$

So the posterior PDF is a combination of three normal distributions. In this example, we are going to sample from this function using the Random Walk Metropolis algorithm defined in definition 2.1.1. Lets run the sampler for 200 iterations and see what happens.

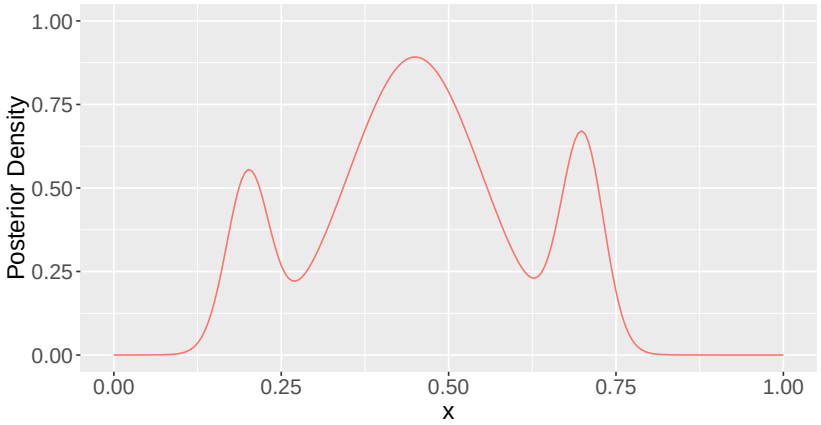


Figure 2.1: Plot of the exact posterior we want the sampler to estimate. We will use this to comment on the accuracy of the Random Walk Metropolis algorithm.

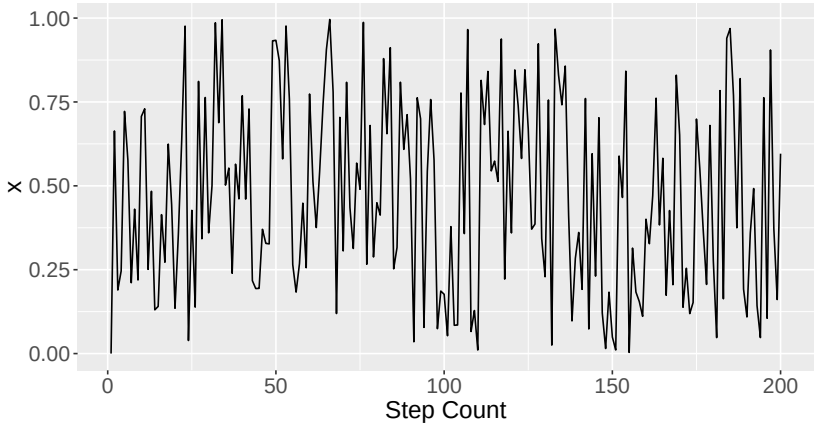


Figure 2.2: Plot of how the parameter x randomly "walks" through parameter space in each iteration of the algorithm.

Figure 2.2 shows how the algorithm moves through different values for the parameter x by accepting the proposed values over 200 iterations. Because each $(n+1)$-th value depended on the n-th value we can track the accepted step $(n+1)$ and link it to the previous step n with a straight line to visualise how each accepted proposed candidate is dependent from the previous value. This produces an entire web presented in Figure 2.3.

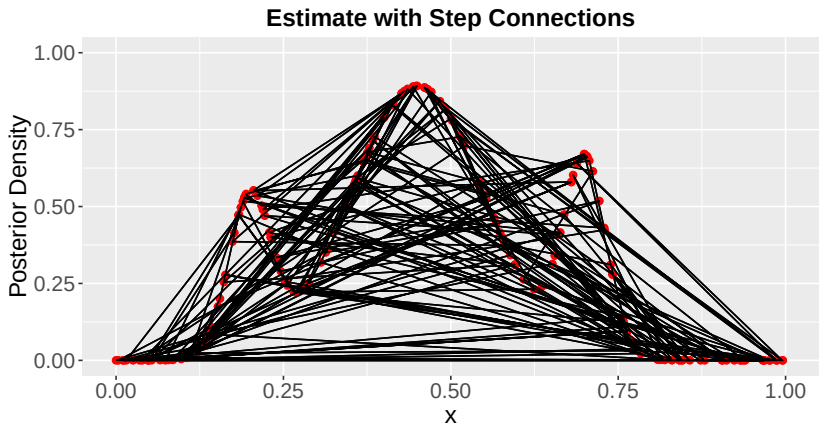


Figure 2.3: Plot of how each point of the posterior estimate is connected to the previous point.

We can then strip away the connections and plot a line through each of the points to determine the accuracy of the sampler given in Figure 2.4.

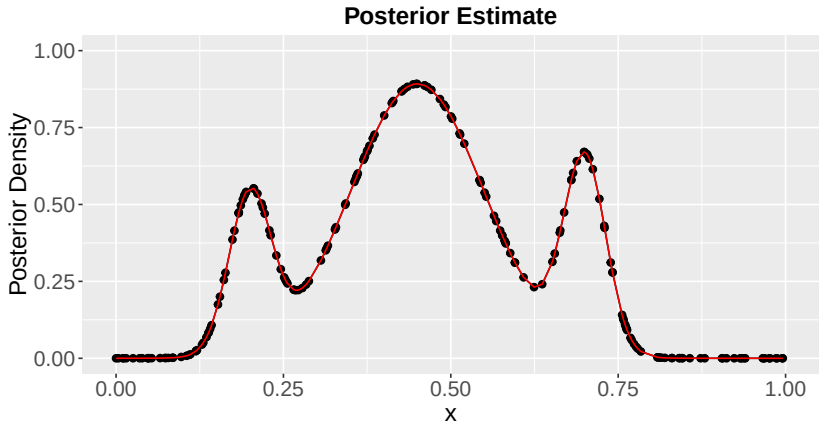


Figure 2.4: Plot of the posterior approximation after 200 iterations without connections.

The estimation is very accurate to the exact graph. In a more real world application, we will not have the posterior as an exact graph. We will want to produce a posterior distribution for the parameter(s) that will then be used to generate a model from some data.

Using a continuous graph helps illustrate more clearly how the algorithm moves. We can now try applying a similar MCMC sampler to some data in another example.

(For the code used in this example visit this link.)

2.2 Metropolis Hastings

Lets take a brief moment to discuss the difference between Metropolis-Hastings and Metropolis sampling algorithms. It was mentioned in definition 2.1.1 that the Metropolis algorithm must have a symmetric proposal distribution. If there are boundary conditions in parameter values, say at zero, then a symmetric distribution can produce a negative value for the standard deviation σ. So, if we then reject all steps where $\sigma < 0$, we will generate few samples from parameter space where σ is close to zero.

Thinking about this more, the areas near zero are only able to be reached by the positive side of σ. This is very different from higher parameter values which can be reached by both sides of σ, meaning that we produce too few samples near zero. So the Metropolis algorithm does not work when there are boundaries in parameter values, like we will have in next example.

Taking a look at Figure 2.5 shows what we mean by this negative σ. Say we have accepted a sample at parameter value 0.04, then we would take this value as the mean of a normal proposal distribution and generate a new sample candidate. The problem comes here when we have a boundary constraint at zero. All the values that would be given in the shaded region would be rejected and we would keep stepping back to 0.04 until we generate a candidate in the positive parameter space. This is what causes asymmetry in the sampler which Metropolis cannot handle.

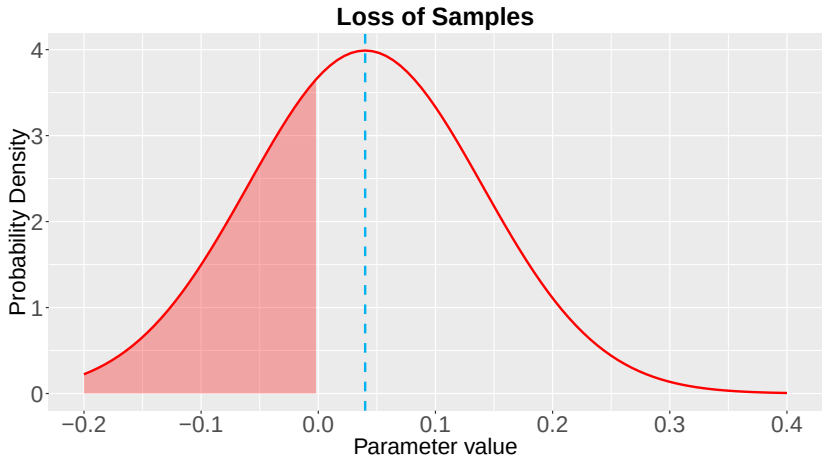


Figure 2.5: Plot of a symmetric proposal distribution, shown using a red line, with a shaded region that shows all of the potential samples we would lose due to a boundary constraint at zero. The blue dashed line is the mean of the proposal distribution; which would be the current parameter value in the sampling algorithm.

So we need to change the sampler and it turns out that we have been using a special case of another sampler. The Metropolis-Hastings algorithm is a modified Metropolis algorithm that allows an asymmetric proposal distribution. And that when we choose a symmetric proposal distribution, it resorts back to the Metropolis algorithm.

An asymmetric proposal distribution will always produce parameter values within the boundary of parameter space i.e. only positive values. Lets see how this algorithm is different.

Definition 2.2.1. ***Metropolis-Hastings:***

1. *Pick a random initial state x_0.*
2. *For each iteration t:*
   - *Randomly generate a candidate state x' from proposal distribution $Q(x' \mid x_t)$.*
   - *Calculate the acceptance ratio*

$$A(x', x_t) = \min\left\{1, \frac{\big(p(data \mid x') \cdot p(x')\big)}{\big(p(data \mid x_t) \cdot p(x_t)\big)} \cdot \frac{Q(x_t \mid x')}{Q(x' \mid x_t)}\right\}$$

   - *Accept of reject:*
     - *Generate a uniform random number $u \in [0, 1]$.*
     - *If $u \leq A(x', x_t)$, accept the new state: $x_{t+1} = x'$.*
     - *If $u > A(x', x_t)$, reject the new state: $x_{t+1} = x_t$.*

Notice how the first part of acceptance ratio is the same as the Metropolis and the second is the correction for the asymmetric stepping. When the proposal distribution is symmetric we have $Q(x_t \mid x') = Q(x' \mid x_t)$ which would then cancel out the right term leaving us with the special case of the Metropolis-Hastings algorithm known as just Metropolis. Now we can be sure that no asymmetries will prevent convergence to the un-normalised posterior.

2.2.1 The Lifespan of a Monster

Suppose that we wanted to research the behaviour of malaria carrying mosquitoes. The most important factor is to estimate the lifespan of a mosquito since the longer it lives, the greater chance it has of becoming infected through biting a human with malaria; surviving the period where the malaria parasite migrates to the salivary glands; and passing the disease on through biting an uninfected human.

One method we can use to collect data is to release 1000 young mosquitoes where each one has been marked. On each day t we use a large number of traps to capture mosquitoes and count the number of marked mosquitoes X_t. We then release all mosquitoes and repeat the process for 15 days in total.

Since X_t is a count variable and we assume that the recapture of an individual marked mosquito is independent and identically distributed, we can choose a Poisson model

$$X_t \sim \text{Poisson}(\lambda_t), \quad \lambda_t = 1000e^{-\mu t}\psi \tag{2.2}$$

Here λ_t is the mean, μ is the mortality rate (assume constant), and ψ is the daily recapture probability. We can use a prior Gamma distribution $\Gamma(2, 20)$ for μ and a prior Beta distribution $\text{Beta}(2, 40)$ for ψ. For a Gamma distribution the mean is $2/20 = 0.1$, and for the Beta distribution the mean is $2/(2 + 40) \approx 0.05$. Substituting this into the equation for λ_t and plotting the result along time t we have the following figure.

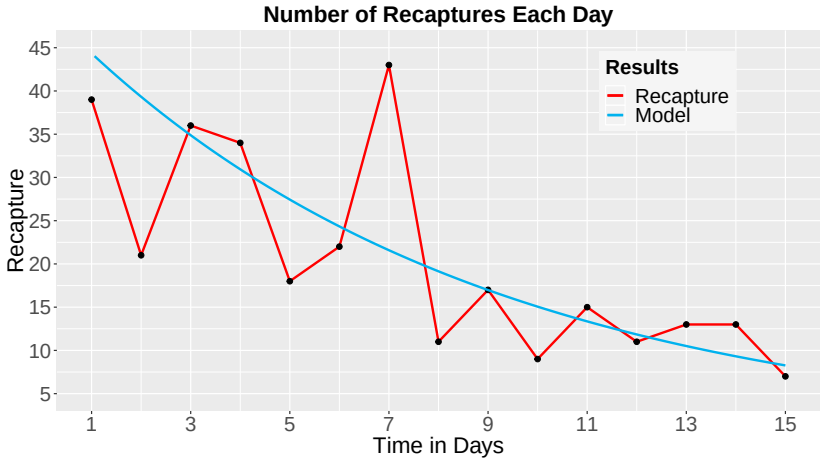


Figure 2.6: Plot of the recapture data using a red line and the Poisson model with mean $\lambda_t = 1000 \times \exp(-0.1 \times t) \times 0.05$ using a blue line.

We have two prior distributions for the two parameters and a likelihood of the Poisson distribution. Then taking the product of these will give us the numerator of Bayes' Theorem which we know is proportional to the posterior.

We still need to propose a proposal distribution so we can calculate the acceptance ratio. We can use a log-Normal proposal with mean at the current μ value, and a Beta$(2+\psi, 40+\psi)$ proposal for ψ. These distributions are asymmetric and so will always be positive, making sure we have no boundary issues.

Now we are ready to substitute this into the Metropolis-Hastings algorithm and record the result. Figure 2.7 is a collection of graphs showing different information we can extract from the algorithm.

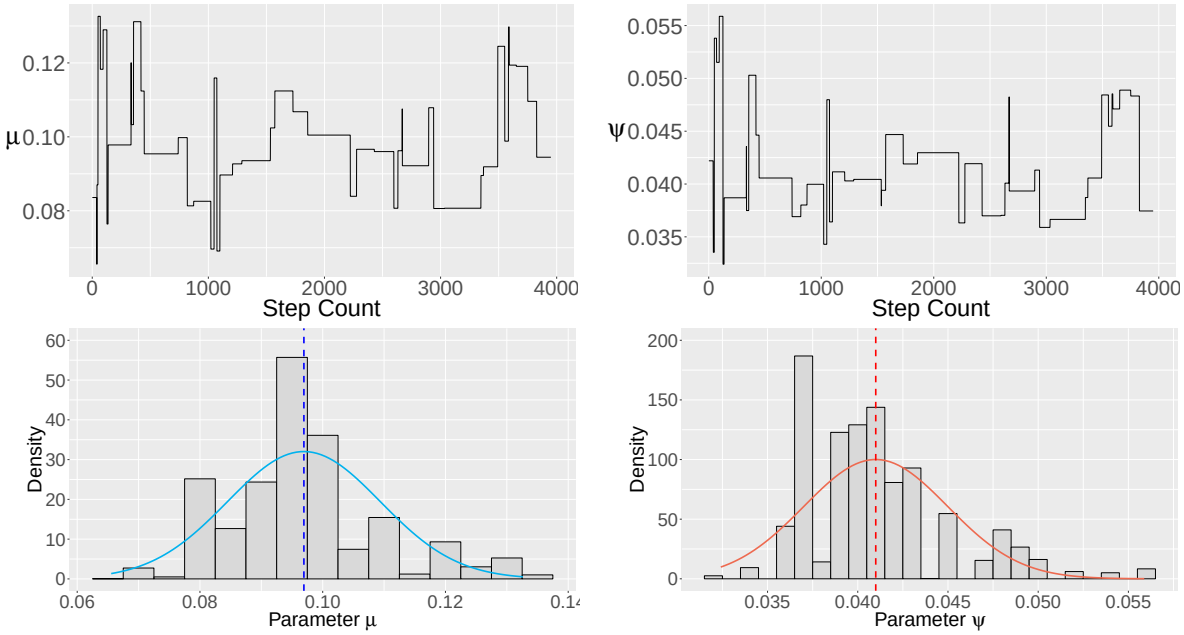


Figure 2.7: The two graphs at the top show the values that parameters μ and ψ take with each iteration. The "flat" sections are for when many potential samples were rejected and the algorithm returned to the previous sample. The bottom graphs show the histogram along with the probability density function and the mean value for parameters μ and ψ. PDFs are plotted using solid lines and means are plotted using dashed lines.

These two histograms represent the estimated posterior distributions for the two parameters. This is the finished product of the Metropolis-Hastings algorithm but we can take it one step further and show what these posterior distributions *really* represent in this real world problem. After running the sampler with 4000 iteration and removing the first 50 values, as these are the values used to converge, we have the parameter mean estimates

$$\mu = 0.096937 \pm \sigma_\mu, \quad \sigma_\mu = 0.012458$$
$$\psi = 0.040997 \pm \sigma_\psi, \quad \sigma_\psi = 0.003988$$

where $\sigma_{\mu,\psi}$ is the standard deviation for each parameter. So the algorithm has generated two posterior distributions, each with their own mean and standard deviation which can be through of as a **confidence interval**. Plus and minus one standard deviation is equivalent to saying that I am 68.2% confident that the parameter values lie within the intervals

$$0.084479 \leq \mu \leq 0.109395$$
$$0.037009 \leq \psi \leq 0.044985$$

The idea of confidence intervals is shown in the Figure 2.8. Here we see how the two parameters are constrained i.e as μ increases so too does ψ. We plot these constraints using all of the accepted parameter values from the 4000 iterations.

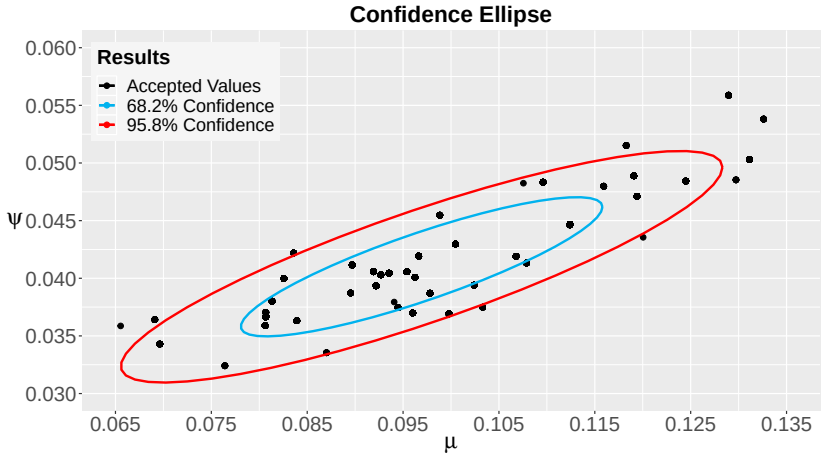


Figure 2.8: Plot of the confidence ellipse for 68.2% which is equal to 1 standard deviation in blue and 95.8% which is equal to 2 standard deviations in red.

This might not look like much but this is the problem finished! We have used only the data along with different prior, likelihood and proposal distributions to produce an estimated value for the mortality rate μ and the daily recapture probability ψ. Now we can uses all of these accepted samples to plot the original Poisson model along with the original data shown in Figure 2.9.

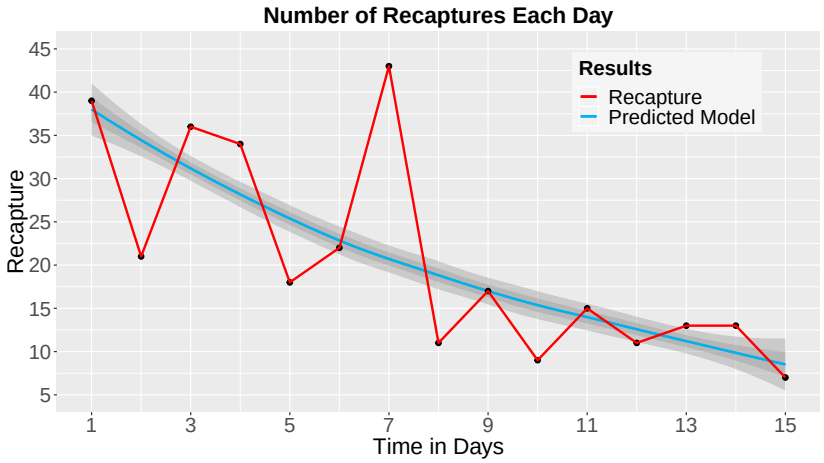


Figure 2.9: Final plot of the model for estimating mosquito lifetime using the accepted values shown in Figure 2.8. The best fit line is plotted with the mean values of μ and ψ using a blue line. The shaded region shows the confidence intervals. The darker region is a 68.2% confidence interval ($1\sigma_{\mu,\psi}$) and the lighter region is a 95.8% confidence interval ($2\sigma_{\mu,\psi}$)

(For the code used in this example visit this link.)

(See [3] Chapter 13 for more detail on §2.1 and §2.2)

Chapter 3

Gibbs & Bayesian Networks

After all of that you might be thinking "this is great!" and you would not be wrong, but is there something better? The main goal for these algorithms is not just to predict a posterior, but how fast they *converge* to the posterior as well. Here we found a good estimate for the parameters but after 4000 iterations which could be quite demanding for weaker hardware. For this reason I will introduce another MCMC sampling algorithm that has the potential to converge much faster, but it too comes with it own disadvantages.

The Metropolis algorithms are great MCMC samplers because of their simplicity but there is the requirement to fine tune the proposal distribution for each new model. They are also **rejection** algorithm which means we lose many potential steps and limits the algorithms ability to explore parameter space quickly.

This is where Gibbs sampler is different. It accepts every proposal step in parameter space. But how is this possible without being able to independently sample from the posterior distribution? The trick is that we can calculate the **conditional** distribution of the parameters meaning that, if we can independently sample from these conditional distributions, then we can accept all proposed steps, thus increasing the rate of convergence.

Though it may not be possible in most cases to find, and independently sample from, the conditional distributions for all parameters, we should be able to for some. We would then use Gibbs sampler for these parameters and a Metropolis-like algorithm for the remaining parameters.

3.1 The Gibbs Sampler

It may seem like Gibbs is better than Metropolis in every way so we should list the potential problems. Firstly, we must be able to calculate conditional distributions which might not be possible at all. Secondly, Gibbs can, in fact, be slow to explore parameter space when there is a strong correlation between parameters.

The previous samplers have functioned by proposing a new point and moving there if the posterior probability is greater than the previous point, or based on a probability which depends on the ratio of the new and old points posterior probabilities. Figure 3.1

shows how the Metropolis sampler moves through parameter space for both parameters. The Gibbs sampler behaves very differently which we will see in the next example. For now, we'll explore how the Gibbs algorithm works.

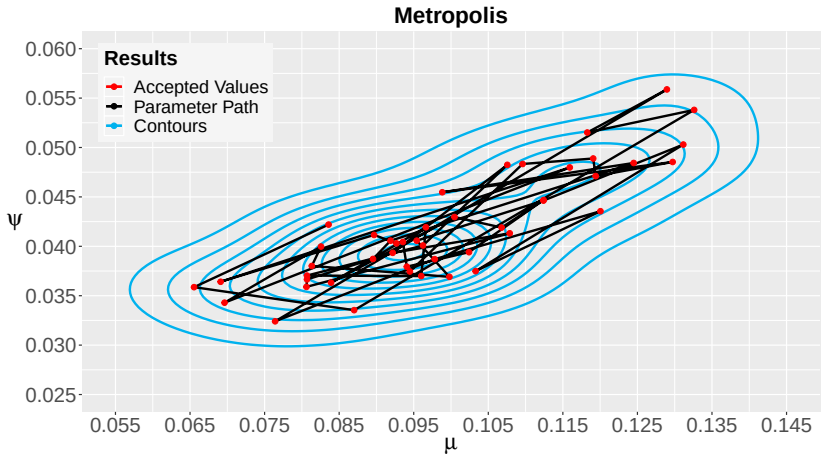


Figure 3.1: This graph shows the path between all of the accepted parameter values for the Metropolis-Hastings algorithm. Notice how, after 4000 iteration, only 46 values were accepted. Though here we have converged to a good estimate in this run, it shows how the high rate of rejection can impede the algorithms ability to converge. Many more iterations are required to increase the reliability of the algorithm.

An important note before we begin is that Gibbs sampling is only *valid* for models with two or more parameters. So to define Gibbs we will use three.

Definition 3.1.1. ***Gibbs:*** *Say we have a model with three parameters* $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$ *and data sample* $\boldsymbol{\gamma}$*. Assume that we can analytically calculate the three conditional distributions given by:*

$$P(\theta_1 \mid \theta_2, \theta_3, \boldsymbol{\gamma}), \quad P(\theta_2 \mid \theta_1, \theta_3, \boldsymbol{\gamma}), \quad P(\theta_3 \mid \theta_1, \theta_2, \boldsymbol{\gamma})$$

Now we do the following

1. *Choose a random starting point* $(\theta_1^0, \theta_2^0, \theta_3^0)$*.*
2. *Choose a random parameter update ordering, say* $(\theta_3, \theta_2, \theta_1)$*.*
3. *Sample from the conditional distribution for each parameter using the most recent parameter value. So, for* $(\theta_3, \theta_2, \theta_1)$*, we would independently sample from*

$$P(\theta_3^1 \mid \theta_1^0, \theta_2^0, \boldsymbol{\gamma}) \to P(\theta_2^1 \mid \theta_1^0, \theta_3^1, \boldsymbol{\gamma}) \to P(\theta_1^1 \mid \theta_2^1, \theta_3^1, \boldsymbol{\gamma})$$

This is repeated until the sampler has converged.

Since the Gibbs sampler functions on conditional distributions, it works well when paired with models known as **Bayesian networks**. These are a probabilistic graphical model that uses Bayesian inference for computations. They model joint probability distributions through conditional independence. Before we begin with Bayesian networks consider the following example.

3.1.1 Crime & Unemployment

Say that the posterior distribution for the unemployment u and crime levels c for a city is estimated to be a bivariate normal distribution which is a normal distribution generalised to two dimensions. Because we can sample independently from the bivariate normal distribution we don't actually need to perform MCMC on this problem but it will be very useful to show in more detail the movement of Gibbs sampler.

For a bivariate normal distribution, the conditional distributions are simple and have the nature of a normal distribution. So, performing a similar process as in definition 3.1.1, we will first choose a random initial state for u^0 and c^0. Then randomly choose which state to update, say u^0. And finally, sample from the conditional distributions with $u^1 \mid c^0$. If the next iteration chose to update c^0 then we would sample from the conditional density $c^1 \mid u^1$. Repeating this process multiple times gives us the following figure.

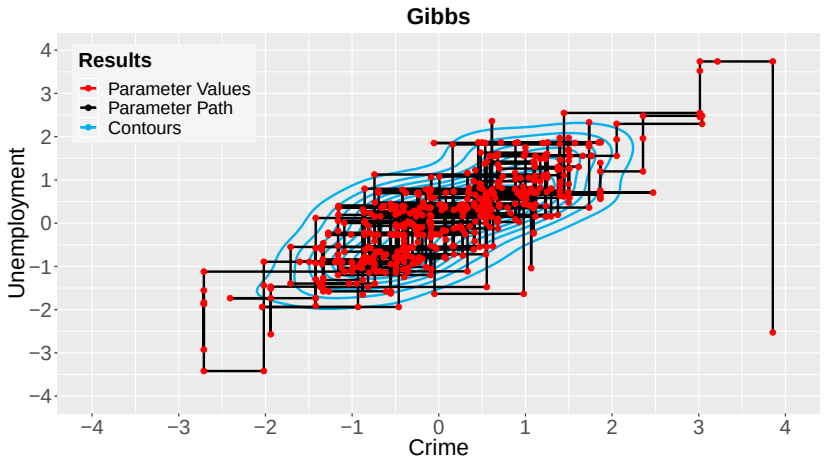


Figure 3.2: This graph shows the path for all parameter values of the Gibbs algorithm. Here there is no reject or accept condition like the Metropolis algorithm. All the parameter values are accepted allowing for faster convergence. This was produced using 500 iterations, far less than the Metropolis algorithms would need to converge.

If you look back at Figure 3.1 you see how the nature of the Metropolis-Hastings algorithm is different to the Gibbs algorithm. Because Gibbs chooses to update one parameter with each iteration this produces a rectangular relation between the points. But, when there is a very strong correlation between the two distributions, Gibbs sampler can struggle to explore parameter space and we might need to resort back to the Metropolis algorithm. It goes to show that only using one algorithm for every problem is not wise. Understanding the features of the problem can aid us to choose the sampling algorithm best suited for those features.

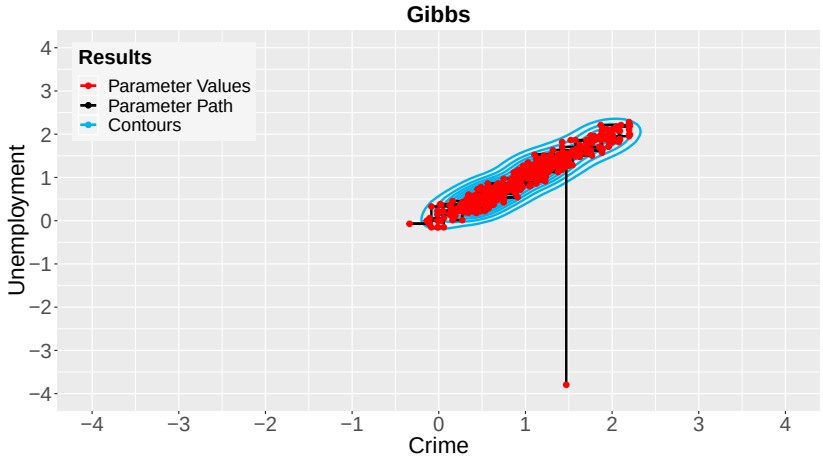
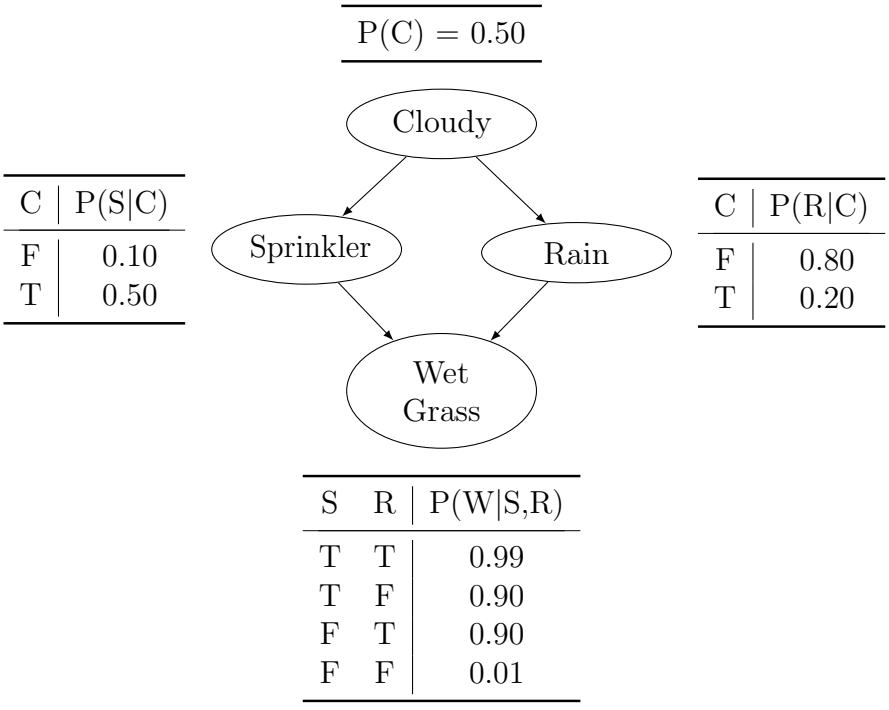


Figure 3.3: This graph shows how the Gibbs algorithm struggles to explore parameter space when there is a strong correlation between the parameter.

(For the code used in this example visit this link.)

Now that we have a good understanding for how the Gibbs sampler is different from the Metropolis samplers we can try an in depth example using Bayesian networks mentioned after definition 3.1.1. Consider the following figure



C	P(S|C)
F	0.10
T	0.50

C	P(R|C)
F	0.80
T	0.20

S	R	P(W|S,R)
T	T	0.99
T	F	0.90
F	T	0.90
F	F	0.01

Figure 3.4: A Bayesian network for why the grass is wet.

Here 'Cloudy' means that the night was completely overcast. We can use Gibbs algorithm to give estimates to **unconditional** probabilities using known conditional probabilities from the Bayesian network. Lets try this out and see what happens.

3.1.2 Why Is the Grass Wet?

Suppose that the grass is wet this morning and I know that the sprinklers when on last night because I stayed up late writing this report. Now, I want to determine (a) the unconditional probability that it was cloudy, $P(C)$, and (b) the unconditional probability that it rained, $P(R)$. In order to do this we will need to calculate the conditional probabilities involved. Some are trivial and some are not but the important thing that was mentioned before is that they are not impossible to calculate. I will go through them now.

- The probability that it was cloudy last night conditional on that it rained, the sprinkler being on, and the grass being wet: $P(C \mid R, S, W) = 0.444$.
- The probability that it rained last night conditional on that it was cloudy, the sprinkler being on, and the grass being wet: $P(R \mid C, S, W) = 0.815$.
- The probability that it was cloudy last night conditional on that it did not rain, the sprinkler being on, and the grass being wet: $P(C \mid \neg R, S, W) = 0.048$.
- The probability that it rained last night conditional it was not cloudy, the sprinkler being on, and the grass being wet: $P(R \mid \neg C, S, W) = 0.216$.

Using these numbers we can implement the Gibbs algorithm and we can tackle both parts (a) and (b) at once. To do this we will use a binomial distribution for the two probabilities $P(C)$ and $P(R)$ since we can only have two outcomes of yes or no (True or False). Firstly, we can construct a function that calculates the different conditional probabilities that we have above. This would look something like:

Pseudocode 3.1: This function is working like a truth table where we have two variable Cloudy and Rained, and the 4 possible truth states. I used the slightly shorter notation, i.e. $P(C \mid R)$ rather than $P(C \mid R, S, W)$, because in this problem we know that the sprinklers went on and that the grass is wet. So for all four states listed above, S and W are always true so we can drop them out as long as we remember why.

```
if(Cloudy = True){
  if(R = True){
    Probability = 0.444. (= P(C | R))
  }
  else{
    Probability = 0.048. (= P(C | ¬R))
  }
}
if(Rained = True){
  if(C = True){
    Probability = 0.815  (= P(R | C))
  }
  else{
    Probability = 0.216 (= P(R | ¬C))
  }
}
return(Probability)
```

Next is to decide which state to update. This would be the function that either makes Cloudy True in line 1 or Rained True in line 9. This is done using a random sample from the binomial distribution mentioned earlier where each state has an equal chance of being selected.

Pseudocode 3.2: This is how we will decide which state to update, either the state Cloudy or the state Rained.

```
18 if(rbinom(1, 1, 0.5) = 1){ (This will only give a value of 0 or 1)
19   Cloudy = True
20 }
21 else{
22   Rained = True
23 }
```

So to recap, say we have that Cloudy is True from line 19, then we would have that Probability = 0.444 from line 3 because R is True in this example. We are looking for $P(C)$ and $P(R)$ so it is defined that C^0 and R^0 are True. Here the superscript denotes the iteration and zero is the initial. Now that we have which state to update we need to update that state.

The value to update the state to, let us call it NewState, is calculated by using the binomial distribution as before, only this time the probability of choosing NewState to be True or False is no longer fair (0.5) but instead the conditional probability that was decided earlier. If we carry on with this hypothetical run, we had that Probability = 0.444 so the probability of success (NewState = True) is 0.444. So it would look something like

Pseudocode 3.3: This is how we will decide what to update the chosen state to, where the probability of success is dependent on the value of the variable Probability.

```
24 if(rbinom(1, 1, Probability) = 1){ (This will give a value of 0 or 1)
25   NewState = True
26 }
27 else{
28   NewState = False
29 }
```

Since Probability is 0.444 it is slightly more likely that NewState = False. If NewState = False for this run then we update the state C^1 = False. This is the crucial aspect to The Gibbs algorithm. We are not rejecting this step. We had that the initial state was C^0 = True. We performed a test run of the algorithm and received the updated state C^1 = False. So, to start the Gibbs sampler, we would have that the first element in the vectors C and R are

$$\mathrm{C} = (\mathrm{C}^0) = (\mathrm{True})$$
$$\mathrm{R} = (\mathrm{R}^0) = (\mathrm{True})$$

and after the test first run of the sampler, we would have the second updated states

$$\mathrm{C} = (\mathrm{C}^0, \mathrm{C}^1) = (\mathrm{True}, \mathrm{False})$$
$$\mathrm{R} = (\mathrm{R}^0, \mathrm{R}^1) = (\mathrm{True}, \mathrm{True})$$

Since $C^1 = \text{True}$ from line 19 we know that C is going to be updated. This means that we make the updated state for $R^1 = R^0$, which is why R is True twice. This would then be repeated again and again where the number of elements in the vector would be the number of times we iterate the sampler.

After all of this explanation we finally have a Gibbs sampler that is working. Running the sampler over 10,000 iteration produces the following histogram results.

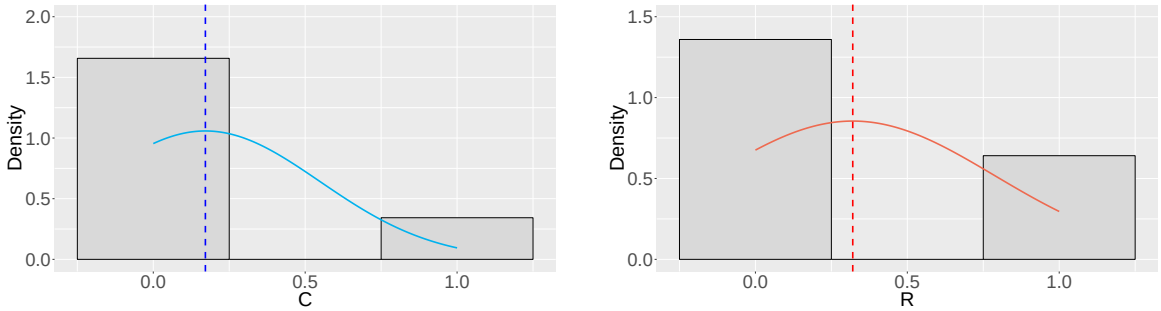


Figure 3.5: These two graphs show the histogram along with thePDF and the mean value for parameters C and R. The PDFs are plotted using solid lines and the means are plotted using dashed lines. Since they only take values True and False we must convert these to numerical values 1 and 0. This is the reason for there being only two levels of frequency.

These are the two posterior distributions for the probabilities $P(C)$ and $P(R)$. So, after only using the conditional probabilities, we have managed to apply Gibbs sampler and arrive at two unconditional probabilities of

$$\begin{aligned} P(C) &= 0.1715 \pm \sigma_C, \quad \sigma_C = 0.3770 \\ P(R) &= 0.3204 \pm \sigma_R, \quad \sigma_R = 0.4667 \end{aligned}$$

where $\sigma_{C,R}$ is the standard deviation. We can also give confidence intervals for these results. For a 95.8% confidence interval we have

$$\begin{aligned} -0.5824 \leq & P(C) \leq 0.9254 \\ -0.6129 \leq & P(R) \leq 1.2537 \end{aligned}$$

Though having a probability less than zero or greater than 1 doesn't make much sense. Plotting the posterior distribution in Figure 3.6 looks rather strange. There are only 4 points corresponding to the 4 possible states the two parameters can be. A very simplified version from Figure 3.2. This is also why so many iterations were required because the algorithm cannot converge to a single point in parameter space.

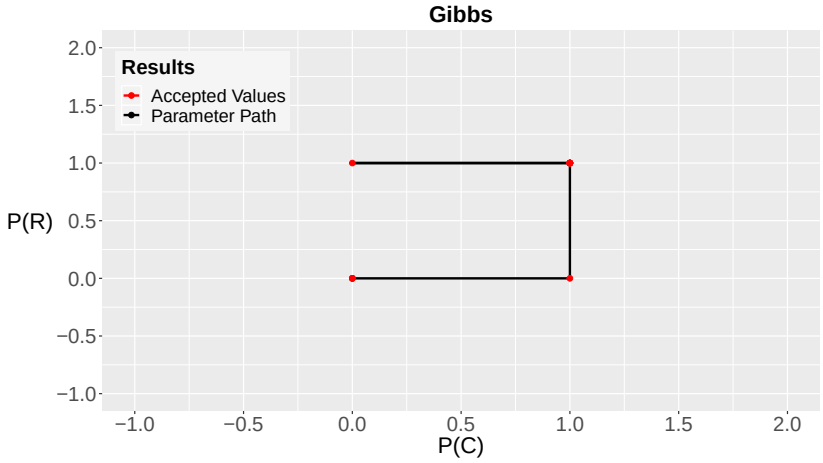


Figure 3.6: Plot of how the Gibbs algorithm moves through parameter space for both parameters. If I were to plot all the accepted values then we just have a unit square so I have restricted the data to the first 15 values. Within these 15 values the sampler moves from (1,1) to (0,1), back to (1,1), then to (1,0), and finally to (0,0).

It is hard to see the movement of the algorithm. The lines connecting each point overlap each other so all movements after the first few are hidden.

(For the code used in this example visit this link.)

(See [3] Chapter 14 for more detail on §3.1)

Chapter 4

Applications of Machine Learning

We move on now from MCMC sampling for an estimate posterior PDF into the more in depth realm of machine learning. I have researched some different machine learning techniques and applied them to some examples. The machine learning process I have focused on is classification. The simplest technique for this is **naïve Bayes**.

4.1 Naïve Bayesian Algorithm

Naïve Bayes Classifiers are a group of probabilistic classifiers that use Bayes' Theorem to make a prediction for an outcome given some features, where these features are assumed to be **strongly independent**. What is a classifier? It is a model that will assign a feature to a class based on observations. But what are features and classes? I'll explain in a moment.

A common application of a naïve Bayes' classifier is binary classification. This means that there are only two classes. For example, we might want to classify if a fruit is an apple or an orange. Here the label would be fruit. A label is just a name given to the group of features, and the classes would be apple or orange. A possible feature for a classifier like this could be the colour of the fruit. So, if we have an orange fruit then we classify it as an orange. The more features we have, the more sophisticated the classifier becomes.

With a basic idea of what a classifier is, we can begin by looking at how this naïve Bayes classification works. Since this classifier used Bayes' Theorem, it is a conditional probability model.

Definition 4.1.1. ***Naïve Bayes:*** *Given a list of labels,* $(x_1, x_2, \ldots, x_n)$ *where* $n \in \mathbb{N}$ *is the number of labels, it assigns the probability*

$$P(C_k \mid x_1, x_2, \ldots, x_n), \quad k \in \mathbb{N} \text{ where } 1 \leq k \leq K$$

Here K *is the number of classes. Using Bayes' Theorem, we can write*

$$P(C_k \mid x_1, x_2, \ldots, x_n) = \frac{P(C_k) \times P(x_1, x_2, \ldots, x_n \mid C_k)}{P(x_1, x_2, \ldots, x_n)} \tag{4.1}$$

The numerator can also be regarded as the joint probability which can be written as

$$P(C_k, x_1, x_2, \ldots, x_n) = P(x_1, x_2, \ldots, x_n, C_k)$$

Using the chain rule this expression can be expanded

$$\begin{aligned} P(x_1, x_2, \ldots, x_n, C_k) &= P(x_1 \mid x_2, \ldots, x_n, C_k) P(x_2, x_3, \ldots, x_n, C_k) \\ &= P(x_1 \mid x_2, \ldots, x_n, C_k) P(x_2 \mid x_3, \ldots, x_n, C_k) P(x_3, x_4, \ldots, x_n, C_k) \\ &\vdots \\ &= P(x_1 \mid x_2, \ldots, x_n, C_k) \cdots P(x_{n-1} \mid x_n, C_k) P(x_n \mid C_k) P(C_k) \end{aligned}$$

Assume that all the labels $(x_1, x_2, \ldots, x_n)$ *are* ***independent****. This gives us*

$$P(x_i \mid x_{i+1}, x_{i+2}, \ldots, x_n, C_k) \approx P(x_i \mid C_k) \;\; \textit{where} \;\; i = 1, 2, \ldots, n$$

This means we can express the joint model as

$$P(C_k \mid x_1, x_2, \ldots, x_n) \approx \frac{P(C_k) \times P(x_1 \mid C_k) \times P(x_2 \mid C_k) \times \cdots \times P(x_n \mid C_k)}{P(x_1, x_2, \ldots, x_n)}$$

Since the denominator does not depend on the class C *it will not change and can be considered a constant. If the denominator is omitted this gives us the following proportionality relation*

$$P(C_k \mid x_1, x_2, \ldots, x_n) \propto P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k) \tag{4.2}$$

We have derived a final expression for the conditional probability model to be used in the classifier. The final stage is to use this expression to define a decision rule, which will decide the class C_k *that the labels* $(x_1, x_2, \ldots, x_n)$ *belong to. This is know as the* ***maximum a posteriori*** *which works by assigning the label to the class where the conditional probability product,* $P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)$ *is greatest. This is given by*

$$\hat{C} = \underset{k \in \{1, \ldots, K\}}{\arg\max} \, P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)$$

The reason why this classifier has its name is due to the assumption we made that all the labels and features are **independent**. In the real world this is not always the case. Some might not be independent, making the final answer *naïve*. But, when applied to the right problem, the naïve Bayes classifier still performs well.

4.2 Binary Classification

Here we have the number of classes $K = 2$. This is most common with scenarios where the answer is yes or no. We will apply the definition of naïve Bayes to an example of binary classification.

4.2.1 To Play or not to Play

Suppose we want to classify if a baseball team will play, depending on the weather. So the label would be *Weather* comprised of the features *Sunny*, *Overcast* and *Rain*, and the classes are *Yes* (the team played) and *No* (the team did not play). We have some observational data for the past few days where the weather was recorded and if the team played baseball, which will then be used to calculate the conditional probabilities needed to perform a naïve Bayes classification.

A keen observation can be made here. Since we only have one label *Weather*, equation (4.1) breaks down to the ordinary Bayes' Theorem we all know and love. Making these calculations the simplest ones possible for a naïve Bayes classification. For the data used in this example see A.1 of Appendix A. So now we can create problems for the classifier to process.

For example, will the team play if it is raining? We need to calculate the probability of the classes given that it is raining. So, the probability that the team will play (class Yes) is given by

$$P(\text{Yes} \mid \text{Rain}) = \frac{P(\text{Rain} \mid \text{Yes}) \times P(\text{Yes})}{P(\text{Rain})}$$

To calculate this we need

$$P(\text{Rain}) = \frac{5}{14} = 0.3571, \quad P(\text{Yes}) = \frac{9}{14} = 0.6429, \quad P(\text{Rain} \mid \text{Yes}) = \frac{2}{9} = 0.2222$$

Substituting this into the equation gives

$$P(\text{Yes} \mid \text{Rain}) = \frac{0.2222 \times 0.6429}{0.3571} = 0.4$$

We then perform the same process for the probability that the team will not play (class No) resulting in

$$P(\text{No} \mid \text{Rain}) = \frac{P(\text{Rain} \mid \text{No}) \times P(\text{No})}{P(\text{Rain})} = \frac{0.6 \times 0.3571}{0.3571} = 0.6$$

So we have that

$$P(\text{No} \mid \text{Rain}) > P(\text{Yes} \mid \text{Rain})$$

Thus, the decision rule would choose the class No. In plain English this means that if it is raining then the team will not play ball. When I first saw this, I was amazed. By only using observed data and Bayes' Theorem, we can predict what the outcome will be for a given condition.

Notice how the denominator is the same in both classes? This is what was mentioned at the end of definition 4.1.1. It is not dependent on the current class so it will remain the same for all classes. Like before, it is just a normalising term that makes the probabilities add up to 1. If we had of omitted the denominator term in both then we would still have the same outcome that the team will not play ball as

$$P(\text{Yes} \mid \text{Rain}) \propto 0.2222 \times 0.6429 = 0.1429$$
$$P(\text{No} \mid \text{Rain}) \propto 0.6 \times 0.3571 = 0.2143$$

What we have just done with basic statistics is machine learning! Now granted, this is a basic level but it still counts. To illustrate the more general process we can add more labels to this example.

4.2.2 Is It Too Hot, or Too Cloudy, or Both to Play Ball?

We will be using the same example and problem as in §4.2.1 but with the extra label *Temperature*. So now we have the label *Weather* consisting of the features *Sunny*, *Overcast*, and *Rain*, and the label *Temperature* consisting of the features *Hot*, *Mild*, and *Cool*. For the data used in this example see A.2 of Appendix A.

For example, will the team play if it is overcast and it is hot? To calculate this we will use equation (4.2) now that we have two labels ($n = 2$). So, the probability that the team will play is given by

$$\begin{aligned} P(\text{Yes} \mid \text{Overcast}, \text{Hot}) &\propto P(\text{Yes}) \times P(\text{Overcast}, \text{Hot} \mid \text{Yes}) \\ &= P(\text{Yes}) \times P(\text{Overcast} \mid \text{Yes}) \times P(\text{Hot} \mid \text{Yes}) \\ &= 0.6429 \times 0.4444 \times 0.2222 = 0.0635 \end{aligned}$$

and the probability that the team will not play is given by

$$\begin{aligned} P(\text{No} \mid \text{Overcast}, \text{Hot}) &\propto P(\text{No}) \times P(\text{Overcast}, \text{Hot} \mid \text{No}) \\ &= P(\text{No}) \times P(\text{Overcast} \mid \text{No}) \times P(\text{Hot} \mid \text{No}) \\ &= 0.3571 \times 0 \times 0.4 = 0 \end{aligned}$$

So we have that

$$P(\text{Yes} \mid \text{Overcast}, \text{Hot}) > P(\text{No} \mid \text{Overcast}, \text{Hot})$$

So this time the decision rule predicts the team will play ball! Hopefully it is not too hot. If it is then machine learning computer does not care. Now we can look into how a process like naïve Bayes is implemented through a computer. If we had dozens of labels then doing this by hand would not be difficult but extremely tedious. That is where Python comes into play.

4.2.3 Scikit-Learn

Scikit-learn have a range of unsupervised and supervised machine learning algorithms all bundled into one Python package. This makes it the perfect tool to use for the machine learning purposes I will cover. We can use the same data from section §4.2.2 and place it into the Scikit-learn process.

The main challenge with machine learning is not the calculations but preparing the data. Here we have data given in words and for scikit to process this we need to convert them

to numbers. This can be done using function called `label encoder`. Running this for the two labels `Weather` and `Temperature`, and the class `Play` we have the following

```
    Weather:  [2 2 0 1 1 1 0 2 2 1 2 0 0 1], 0:Overcast, 1:Rain, 2:Sunny
Temperature:  [1 1 1 2 0 0 0 2 0 2 2 2 1 2], 0:Cool, 1:Hot, 2:Mild
       Play:  [0 0 1 1 1 0 1 0 1 1 1 1 1 0], 0:No, 1:Yes
```

Looking back at A.2 we see what each number corresponds to in each label and class.

Next is to link the features of `Weather` and `Temperature` together into one variable, resulting in a list of tuples. Even though the choosing of which number goes to which feature is done alphabetically, the original order of the data is preserved. So we can just group up the two as they are. I have chosen to call this variable `Label`

```
Label:  [(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0),
         (2, 2), (2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]
```

Lastly, we are ready in perform naïve Bayes classification, just as we did before by hand, through Scikit. One technical note, Scikit naïve Bayes classifiers need to be given an assumption about how $P(x_i \mid C_k)$ is distributed in equation (4.2). So to get this classifier working we will choose the most basic *Gaussian* naïve Bayes given as

$$P(x_i \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_{C_k}^2}} \cdot \exp\left(-\frac{(x_i - \mu_{C_k})^2}{2\sigma_{C_k}^2}\right)$$

where $\sigma_{C_k}^2$ and μ_{C_k} are estimated using maximum likelihood. So, for this example, the likelihood of the labels is assumed to be normally distributed. It's just an assumption we have made that won't affect the process very much. The code is simple to write, using the function `GaussianNB`, as

```
model = GaussianNB()
model.fit(Label,Play)
predicted = model.predict([[0,1]])
```

where `0:Overcast` and `1:Hot`. We grouped the features in the order `Weather` then `Temperature` so that is the order they need to be entered into `model.predict()`. Running this code gives the output

```
predicted = 1
```

which we know for `Play` is a `Yes`. We have the same result we calculated by hand (which is reassuring). Because the dataset is so small (14 observations), the classifier will not be very accurate. If we had more data then the classifier would be more reliable.

This concludes machine learning for binary classification! We gave a machine some data and it made a prediction on the outcome for given features. Furthermore, we now understand *how* the machine used naïve Bayes to calculate the different probabilities and select the greatest one as the prediction.

I would next like to cover an example application of machine learning that is performed for everyone without even realising it. Have you ever opened you emails and seen that you have junk emails? How does your email service know that these emails are junk? The truth is that these emails are being classified using a machine learning technique. They could be using naïve Bayes algorithm or another form of classification. Lets see how a process like this would work.

(For the code used in this example visit this link.)

4.2.4 Politics or Sports

We are going to build a classifier to predict if a sentence is about politics or sports. Suppose we have the following dataset given in A.3 of Appendix A and we want to predict if the sentence "A very close game" is about sports or politics. Obviously this is about sports but can the classifier get it right?

The way to do this is by adding the sentence to the dataset and using the first 5 as training data and the last sentence as testing data. Because we are working with text we need to remove the capitalization so "A" and "a" are not treated as two different words. Then we need to encode this data like before by dividing the sentences into individual words.

There are 14 words in total so each word will be encoded from `0` to `13` and the number of labels is the number of words in the sentence to be classified. This is why we enter the sentence into the dataset as calling it through numbers would be confusing. lastly, we set split the training and test data as

```
N = 6
n = N-1
X_train = Data['Text'][:n]
Y_train = Data['Tag'][:n]
X_test = Data['Text'][n:N]
Y_test = Data['Tag'][n:N]
```

where `N` is the number of sentences. We train up to the 5^{th} sentence and test the last one. To do this example by hand, the probabilities are calculated in the same way as §4.2.2. To find these you would need to exclude the last sentence from the dataset as that's the one we are now working with. It would look like the following

$$\begin{aligned} P(\text{Sports} \mid \text{a, very, close, game}) &\propto P(\text{Sports}) \times P(\text{a, very, close, game} \mid \text{Sports}) \\ &= P(\text{Sports}) \times P(\text{a} \mid \text{Sports}) \times P(\text{very} \mid \text{Sports}) \times \\ &\quad P(\text{close} \mid \text{Sports}) \times P(\text{game} \mid \text{Sports}) \\ &= \frac{3}{5} \times \frac{2}{11} \times \frac{1}{11} \times \frac{0}{11} \times \frac{2}{11} = 0 \end{aligned}$$

This isn't good. The word "close" does not appear in the Sports text, which is giving us a zero result. This means that we are losing all the information from the other probabilities.

The workaround for this issue is known as **Laplace smoothing** where we just add one to every word count so that there is never a zero probability in the calculation. And to balance this increase, we add the total number of words to the denominators so the probabilities will never be greater than 1. Doing this gives

$$\begin{aligned} P(\text{Sports} \mid \text{a, very, close, game}) &\propto \frac{3}{5} \times \frac{2+1}{11+14} \times \frac{1+1}{11+14} \times \frac{0+1}{11+14} \times \frac{2+1}{11+14} \\ &= \frac{3}{5} \times \frac{3}{25} \times \frac{2}{25} \times \frac{1}{25} \times \frac{3}{25} = 2.7648 \times 10^{-5} \end{aligned}$$

Repeating for the class Politics, and using Laplace smoothing again since "very" and "game" are would give zero conditional probabilities, gives

$$\begin{aligned} P(\text{Politics} \mid \text{a, very, close, game}) &\propto P(\text{Politics}) \times P(\text{a, very, close, game} \mid \text{Politics}) \\ &= P(\text{Politics}) \times P(\text{a} \mid \text{Politics}) \times P(\text{very} \mid \text{Politics}) \times \\ &\quad P(\text{close} \mid \text{Politics}) \times P(\text{game} \mid \text{Politics}) \\ &= \frac{2}{5} \times \frac{1+1}{9+14} \times \frac{0+1}{9+14} \times \frac{1+1}{9+14} \times \frac{0+1}{9+14} \\ &= \frac{2}{5} \times \frac{2}{23} \times \frac{1}{23} \times \frac{2}{23} \times \frac{1}{23} = 5.7175 \times 10^{-6} \end{aligned}$$

At last we have the conclusion that

$$P(\text{Sports} \mid \text{a, very, close, game}) > P(\text{Politics} \mid \text{a, very, close, game})$$

so we would choose that the sentence "A very close game" is about Sports. Huzza! We have a correct prediction from the classifier. Running the Python code using scikit for the same sentence gives the result

```
predicted = ['Sports']
```

So we know the classifier in Python is working correctly since we checked the mathematics. Although, there is an interesting issue that depends on the sentence you choose to classify. For example, If we wanted to classify the sentence "Not a violent debate", we know this is about Politics but the only word in this sentence that is in the training data is "a". Even with the Laplace smoothing, we would have that

$$\begin{aligned} P(\text{Sports} \mid \text{not, a, violent, debate}) &\propto P(\text{Sports}) \times P(\text{a} \mid \text{Sports}) \times \cdots \\ &= \frac{3}{5} \times \frac{3}{25} \times \frac{1}{25^3} = 4.608 \times 10^{-6}, \end{aligned}$$

$$\begin{aligned} P(\text{Politics} \mid \text{not, a, violent, debate}) &\propto P(\text{Politics}) \times P(\text{a} \mid \text{Politics}) \times \cdots \\ &= \frac{2}{5} \times \frac{2}{23} \times \frac{1}{23^3} = 2.859 \times 10^{-6}. \end{aligned}$$

So this is why running the Python code with the sentence "Not a violent debate" is incorrectly classified as

```
predicted = ['Sports']
```

One counter measure to this is removing stopwords such as “a”, “else”, “ever”, words that don’t add information but we would still have the incorrect answer of sports since

$$\frac{3}{5} \times \frac{1}{25^3} = 3.84 \times 10^{-5} > 3.29 \times 10^{-5} = \frac{2}{5} \times \frac{1}{23^3}$$

We simply need to have more data to make the classifier more reliable. Which is why the larger the dataset, the better the machine learning result. We can try some other sentences for fun and see the results from the code. I will put them in a table below

Sentence	Prediction
Voting time is over	`['Politics']`
The election result was tampered with	`['Politics']`
A forgettable election	`['Politics']`
It is a great game	`['Sports']`
That is a touchdown	`['Sports']`
Homerun	`['Sports']`
It is a homerun	`['Politics']`

Adding all of these correct predictions to the dataset would improve its accuracy even more. We were lucky with the last two correct sports predictions as none of the words are in the training data so they were given the class Sport purely from the above calculation.

The last sentence is given the wrong prediction `['Politics']` because of the word “it”. But if we update the training data with “It is a great game” being Sports, then we have

```
It is a homerun = ['Sports']
```

The main flaw is with the dataset. We have that when no words are used in training, we get Sports. A simple fail safe could be added where the program asks for the classification of a sentence to be given when there are no known words.

Please view the Python code that I have written for this example. It took much time and careful programming to achieve. I would like to discuss the inner workings of the code more but feel that it would be too complicated for those who are not familiar with Python. So I will leave a link to the code for viewing on my GitHub repository.

(For the code used in this example visit this link.)

(See [5] for more detail on §4.1 and §4.2)

4.3 Multiple Classification

We have spent time with binary classification so lets take what has been covered here into the more interesting **multiple classification**. This is when the number of classes $K \geq 3$. Taking everything coved through §4.1-§4.2.3, we will tackle another example using Scikit.

4.3.1 Is This Wine Locally Sourced

For this example I am using a famous multi-class classification problem involving wine. The dataset is available at the following link. The data is for wines grown in the dame region in Italy but derived from three different cultivars. This is much more technically challenging because the dataset is much larger than the previous ones so I will only tackle this example using Python.

The dataset is loaded into Python through scikit as

```
from sklearn import datasets
wine = datasets.load_wine()
```

Exploring the dataset will tell us more about the features, labels and classes. Here there is one label `Wine` and 13 different features each with 178 values, and three different classes. We will use naïve Bayes to predict what class a wine belongs to given its 13 different feature values. Lets see how this looks

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash',
            'magnesium', 'total_phenols', 'flavanoids',
            'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity',
            'hue', 'od280/od315_of_diluted_wines', 'proline']
   Class:  ['class_0' 'class_1' 'class_2'
```

That is a lot of different features. Who knew that wine could be so complicated! If we view the first row of the dataset we see that the data is given in decimals so there is no feature encoding needed this time

```
[[ 1.42300000e+01 1.71000000e+00 2.43000000e+00 1.56000000e+01
   1.27000000e+02 2.80000000e+00 3.06000000e+00 2.80000000e-01
   2.29000000e+00 5.64000000e+00 1.04000000e+00 3.92000000e+00
   1.06500000e+03]]
```

The double brackets here are because the data is stored in one vector comprised of 178 vectors each with 13 elements for each feature. The first row is just 1 of the 178 vectors. Now, for each of these vectors, we know which class it belongs to. Just like how we knew if the baseball team played or not for the 14 observations is §4.2.3.

The next crucial step is called **splitting the data**. This is for testing the accuracy of the naïve Bayes classifier. We will have a portion of the dataset for training and the rest for testing. This can all be performed in scikit with a single line of code using the `train_test_split` function

```
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target,
                                   test_size=0.3,random_state=x)
```

where the `X` variables are the feature data and the `y` variables are the class data. The `test_size=0.3` argument means that 30% of the dataset will be used for testing so the

remaining 70% is used for training. `random_state=x` is for a random number generator where, for a positive integer `x`, the random numbers will be the same for consistency in testing so that the classes and the associated feature values are all mixed up. Lets choose `x = 32`. Then, we do the same as we did before

```
model = GaussianNB()
model.fit(X_train,y_train)
predicted = model.predict(X_test)
```

To then calculate the accuracy we use scikit `metrics` module

```
Accuracy = metrics.accuracy_score(y_test, predicted))
         = 0.9814
```

So the Gaussian naïve Bayes predictions were 98.1% correct. But sometimes the classifier would perform better depending on the value of `random_state`. So I performed a loop where the classifier would run 50 times with a different `random_state` value each time going from `x = 1,2,3,...,50` and take the average of all these percentages.

I would then repeat this process for different values of `test_size` and plot the results to visualise how increasing the training size makes the classifier perform better in Figure 4.1. The more data that the classifier has to train with, the *smarter* it becomes. After training size = 40% the machine learning process is not really *learning* any more.

We see the average accuracy percentage start to level out. This is good to note because at training size = 40% the average percentage accuracy is 96.53211%. If this was accurate enough for some hypothetical purposes then we would not need to continue training the classifier, essentially making the process more efficient.

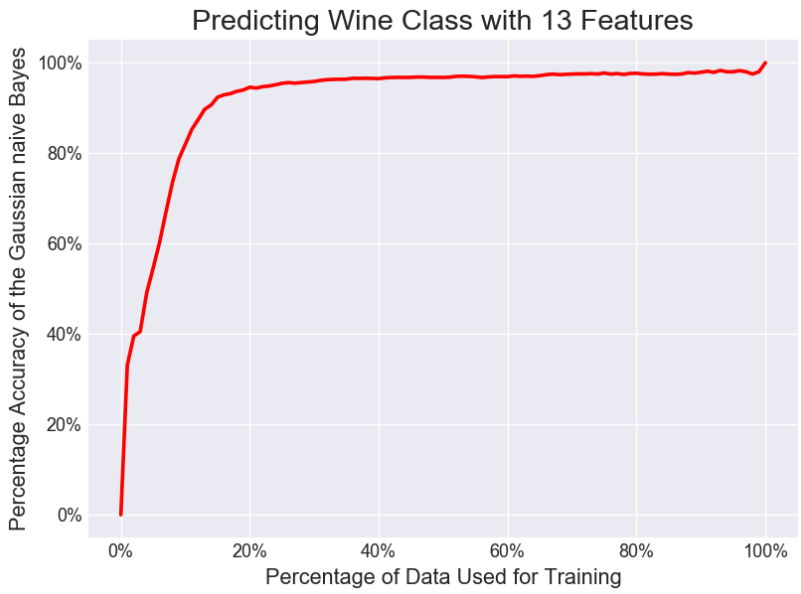


Figure 4.1: Plot of the average accuracy for the Gaussian naïve Bayes classifier for the percentage of dataset used for training. It takes the form of the logarithmic progression. Before even using just 20% of the data as training we already have an average accuracy greater than 80%.

(For the code used in this example visit this link.)

Conclusions

Working with MCMC algorithms and machine learning methods, I have gained the skills to apply statistical mathematics to real world problems. After reading this report you should have a complete understanding on:

- Why performing analytical calculations using Bayes theorem is not always possible and why the posterior is proportional to the un-normalised posterior.
- The Metropolis algorithm and how the acceptance ratio calculates the probability that the next sample value is accepted or rejected.
- The Metropolis-Hastings algorithm and its modification that prevents potential samples being lost due parameter boundaries by allowing asymmetric proposal distributions.
- The Gibbs algorithm and its ability to accept all proposed samples, allowing for faster convergence to the posterior distribution.
- Using a Bayesian network and applying Gibbs algorithm to sample from the different conditional probabilities to give estimates on unconditional probabilities.
- The mathematics of the naïve Bayes classification algorithm and its application in everyday life.
- Why a machine learning algorithms accuracy is proportional the the amount of data it has to learn from.

What I would do given more time is an introduction into the mathematics of neural network. These are used for much more complicated classification such as finding patterns in handwriting. I also wanted to formulate a problem where all sampling algorithms could be applied and compare the rate of convergence for each. Depending on the severity of correlation between the parameters would determine which algorithms would converge the fastest. I will still go on to learn more about neural networks after my project because of my vast interest in the subject. It is the gateway to the future and we must tread with caution.

Acknowledgements

This work would not have been possible without the support of everyone involved.

I am especially indebted to Dr. Siri Chongchitnan, who has worked actively to provide me with guidance and support throughout my four year Master of Mathematics degree.

Thank you to my Mother and Father, Christina Turner and Leonard Turner, for working hard to proofread this report and for always encouraging me to be the best I can be.

Thank you to my Auntie, Diane Greening, for your constant care and being the person in my life who introduced me to the world of science. Enrolling in a mathematics degree might never have happened without your influence.

Thank you to my Partner, Rebecca Baker. We have made it through university only because of the daily love and support we give one another. I will always be thankful to have you in my life.

Thank you to my Sister, Kerri-Marie Turner, for always believed in me growing up.

And finally, thank you to my Grandmother, Kathleen Greening, for taking care of me throughout the years from primary school to university.

Bibliography

[1] Magnus Vilhelm Persson and Luiz Felipe Martins. *Mastering Python Data Analysis*. Packt Publishing Ltd, 2016.

[2] Osvaldo Martin. *Bayesian analysis with Python*. Packt Publishing Ltd, 2016.

[3] Ben Lambert. *A Student's Guide to Bayesian Statistics*. SAGE Publications Ltd, 2018.

[4] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[5] Scikit-learn naive bayes documentation. https://scikit-learn.org/stable/modules/naive_bayes.html.

Appendix A

Tables

Table A.1: Data table for the weather and the play outcome, the frequency table for the weather, the likelihood table for the overall probabilities for weather and play, and the posterior probabilities table for Play. Here "PP" stands for Posterior Probability.

Weather	Play
Sunny	No
Sunny	No
Overcast	Yes
Rain	Yes
Rain	Yes
Rain	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rain	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rain	No

Weather	Yes	No
Overcast	4	0
Sunny	3	2
Rain	2	3
Total	9	5

Weather	Yes	No	Total	Weather Probability
Overcast	4	0	4	$4/14 = 0.2857$
Sunny	3	2	5	$5/14 = 0.3571$
Rain	2	3	5	$5/14 = 0.3571$
Total	9	5	14	
Play Probability	$9/14 = 0.6429$	$5/14 = 0.3571$		

Weather	Yes	No	PP of Yes	PP of No
Overcast	4	0	$4/9 = 0.4444$	$0/5 = 0$
Sunny	3	2	$3/9 = 0.3333$	$2/5 = 0.4$
Rain	2	3	$2/9 = 0.2222$	$3/5 = 0.6$
Total	9	5		

Table A.2: Data table for the weather, temp and play, frequency table for the weather and the temp, the likelihood table for the overall probabilities for weather and play, and temp and play, and the posterior probabilities tables for play for weather and temp.

Weather	Temperature	Play
Sunny	Hot	No
Sunny	Hot	No
Overcast	Hot	Yes
Rain	Mild	Yes
Rain	Cool	Yes
Rain	Cool	No
Overcast	Cool	Yes
Sunny	Mild	No
Sunny	Cool	Yes
Rain	Mild	Yes
Sunny	Mild	Yes
Overcast	Mild	Yes
Overcast	Hot	Yes
Rain	Mild	No

Weather	Yes	No
Overcast	4	0
Sunny	3	2
Rain	2	3
Total	9	5

Temp	Yes	No
Hot	2	2
Mild	4	2
Cool	3	1
Total	9	5

Weather	Yes	No	Total	Weather Probability
Overcast	4	0	4	$4/14 = 0.2857$
Sunny	3	2	5	$5/14 = 0.3571$
Rain	2	3	5	$5/14 = 0.3571$
Total	9	5	14	
Play Probability	$9/14 = 0.6429$	$5/14 = 0.3571$		

Temp	Yes	No	Total	Temp Probability
Hot	4	0	4	$4/14 = 0.2857$
Mild	3	2	6	$6/14 = 0.4286$
Cool	2	3	4	$4/14 = 0.2857$
Total	9	5	14	
Play Probability	$9/14 = 0.6429$	$5/14 = 0.3571$		

Weather	Yes	No	PP of Yes	PP of No
Overcast	4	0	$4/9 = 0.4444$	$0/5 = 0$
Sunny	3	2	$3/9 = 0.3333$	$2/5 = 0.4$
Rain	2	3	$2/9 = 0.2222$	$3/5 = 0.6$
Total	9	5		

Temp	Yes	No	PP of Yes	PP of No
Hot	2	2	$2/9 = 0.2222$	$2/5 = 0.4$
Mild	4	2	$4/9 = 0.4444$	$2/5 = 0.4$
Cool	3	1	$3/9 = 0.3333$	$1/5 = 0.2$
Total	9	5		

Table A.3: Data table for the text and tag, and splitting text and adding test text, and the frequency tables for the text, text in Sports, and text in Politics before adding test text.

Text	Tag
A great game	Sports
The election was over	Politics
Very clean match	Sports
A clean but forgettable game	Sports
It was a close election	Politics

Text	Tag
a, great, game	Sports
the, election, was, over	Politics
very, clean, match	Sports
a, clean, but, forgettable, game	Sports
it, was, a, close, election	Politics
a, very, close, game	Sports

Words	Frequency
a	3
game	2
close	1
clean	2
very	1
election	2
was	2
it	1
but	1
forgettable	1
match	1
the	1
over	1
great	1

Words in Sports	Frequency
game	2
a	2
clean	2
very	1
forgettable	1
but	1
match	1
great	1

Words in Politics	Frequency
was	2
election	2
close	1
it	1
a	1
over	1
the	1