Week 3 meeting notes

|  |
| --- |
| Generative model: some population, we choose mu and sigma and we put it through to get an array x.  xi ~ N(μ,σ)  Each xi is drawn from a normal distribution with mean μ and standard deviation σ  x = np.random.randn(N)\*σ + μ  xobs ~ N(xi, σobs)  xobs = x + np.random.randn(N)\*σobs  np.random.randn is a normal distribution with mean = 0 and standard deviation 1 which then is scaled by being multiplied by sigma (to create noise in the data I think)  we specify a prior on μ  μ ~ N(0,1000)  so μ is normally distributed with a mean of 0 and a standard deviation of 1000  the true value of μ is 1 number, but when you don’t know μ, you say that with some certainty μ will sit comfortably within the distribution.  You want the likelihood to be narrow compared to the prior, so that you’re not biasing the result.  To deal with σ you can do a half normal distribution  σ ~ Half-N(0,1000)  it’s a half normal distribution (can’t go below 0)    θ, Φ are the hyper parameters  Φ and θ are both arrays  θ is each value of x (it can be more but here it is x)  Φ is μ and σ  D is the data (the real obs)  The hyper prior = P(Φ): μ ~ N(0,1000), σ ~ Half-N(0,1000)  Prior = P(D|θ): x ~ N(μ,σ)  P(θ|Φ) = xobs ~ N(x, σobs)  The hyperprior gives the prior  “the hyperprior are the priors on the parameters of the prior”  If you have say 500 points (i.e. N=500 in x = np.random.randn(N)\*σ + μ), then 500 values of x will be created x1, x2…, based off of the hyperprior  Bayes theorem: P(θ, Φ|D) ∝ P(Φ) P(θ|Φ) P(D|θ)  P(D|θ) could here be written as P(D|θ,Φ), however this is unnecessary because θ comes from Φ  (θ is x i.e. the prior which is developed for the hyperprior parameters μ and σ, which is Φ) |
| Import pymc3 as pm  Model = pm.model |
| import pymc3 as pm  import numpy as np  import matplotlib.pyplot as plt  mu=1000  sigma=10  N=1000  sigma\_obs=50  x=np.random.randn(N)\*sigma+mu  x\_obs=x+np.random.randn(N)\*sigma\_obs  model=pm.Model()  with model:  mu = pm.Normal('mu',0,10) #mu is drawn from normal dist(name,mean,sd)  sigma=pm.HalfNormal('sigma', 10)    xtrue = pm.Normal('xtrue', mu, sigma, shape=N) #normal dist(name,mean=mu,sd=sigma)  xobs = pm.Normal('xobs', xtrue, sigma\_obs, observed=x\_obs)  #xtrue = true ages of stars for ‘simulation’ given mu and sigma  #xobs = observational uncertainty = observation with noise  #x\_obs = observations array)  with model:  trace = pm.sample(tune=4000, target\_accept=0.99)  #it does the numerical integration by first learning the shape and scale of the posterior  #distribution. After that it can do effective sampling.  #tune = no. steps  #hpd = highest posterior density  #nf = no. effective samples = from the posterior it believes it has some no. effective samples from #mu  #rhat = Gelman and Rubin metric to say if the algorithm converged, if it’s above about 1.01/1.02 it #probably hasn’t converged properly, so run it for longer.  pm.summary(trace)  for mu, we have a large uncertainty because we don’t know what the true mu is at all  the uncertainty is 1000 and our initial guess is that mu is 0 so when it does the NUTS sample it searches from -1000 to 1000, and if we do N=100 i.e. 100 samples, 64 of them will be within 1 sigma of true mu  How it works:  We give it a guess for mu (the mean age of the stellar population), mu\_sigma (uncertainty on our guess on the age.) If we have data we can take mu and mu\_sigma form the data. However if we don’t have any idea on what the ‘true mu’ is, we make mu\_sigma very large compared to mu. This is because the given mu and mu\_sigma are used to create a normal distribution (with our guess of mu at the peak) where the density of points increases towards the peak of the distribution. As a result the vast size of the uncertainty on mu should allow for the true mu to sit fairly near the peak of the distribution (which is where our guess of mu is by construction).  We also give a sigma i.e. the standard deviation of the ages of the population and a sigma\_sigma i.e. our uncertainty on the standard deviation of the ages, which follow the same rules as above and are used in a HalfNormal distribution because the standard deviation cannot be negative in any situation (as we will be looking at ages, our mu cannot be negative either…but anyway)  We then give some observational data for the stellar population we are interested in but we also include the observational error (sigma\_obs) to add noise to the stellar populations we create based on the above 4 parameters: mu, mu\_sigma, sigma, sigma\_sigma  (the created populations also follow a normal distribution).    Running the sampler:  Step 1: the algorithm does some very clever selection of mus and sigmas from their distributions  The no. selections is dependent on the tuning value given.  For each of the selected mus and sigmas, an xtrue (simulated population) is created with some mui and sigmai.  xobs (simulated population with noise) is then created by adding sigma\_obs into the xtrue data.  This process is of creating simulated populations is repeated a no. mus and sigmas chosen.  These sample populations are then compared to the observational data to see which simulated population best matches the observational data. The best matching simulated population should then have been simulated from a mu and sigma closest to the true mu and true sigma  Step 2:  The algorithm limits it’s field of study to a much smaller field around the mu and sigma of the best simulated population and repeats step 1 (a number of times I do not know i.e. does it create a no. simulated populations = the tuning again and does it refine its field of view multiple times). |
| Sigma\_obs/sqrt(N) = minimum usable resolution |
| Use M67 published ages  Get ages for lots more stars in potentially other clusters, to do that we need models of stellar evolution i.e. grid models  For that interpolation is done but that is slow, so instead we could calculate the tracks for each point (evolves a star to that point and then the sample moves to a new position and evolves again)  Alternatively, machine learning is built on linear algebra (almost exclusively) and is very fast, if we can build a machine learning algorithm that emulates stellar evolution, here is a grid of models, go learn it.  If we take the grid model that spans a region in space that is applicable to a cluster and we tell a neural net to learn (put in an age and mass and out comes observables e.g. delnu and numax)  Numax is currently an unknown sort of, so we will kind of ignore it  We will be getting numax values but they will likely but unreliable so in our final report we should have a disclaimer:  “We’re gonna use numax but we understand that there is no formal theoretical relationship between numax and the models we are calculating therefore we approximate it with this parameter this parameter that was published in this paper[citation needed], this is a major limitation of “the current things” but in future a better calibration and scaling relations… ”  We care about the method: machine learning, hierarchical models  We’re going to do a generative model: “put in fundamental and get observables”  Initially we will mostly ignore the systematic uncertainties that plague astrophysics  If we have enough time we can return to the systematics and try to solve some of those problems  Which essentially allows for endless scope beyond what is required. |
| Universal approximation function = it is mathematically guaranteed that you can approximate any function using a neural net of sufficiently large architecture.  (though it doesn’t mean you can train the neural net) |
| Proposal stuff: what do scientists do when they plan projects  For Abstract:  Context  Aims  Methods Results  Conclusions  Implications  Stellar clusters are really exciting but we don’t know much about them  So we are going to model them to estimate the age spread  Context: other people have done this (looked at clusters and populations) before so mention the paper we found with the ages in it. Other people have used methods like…but not in the same context e.g. there are hierarchical models (see oli’s paper?)  Methods: neural net emulater and hierarchical modelling to find the spread  Results: Hopes and Aspirations,  Gant charts? E.g. on a timescale of this and this we are going to do x,y,z  Put thought into it despite knowing it may become completely obsolete  after proposal submission  Conclusions: if we get a results like this we can get that  Implications: doing this for more clusters would yield better results  Try to detail how we intend to do a better job than the researchers that came before us.  Which comes down to the square root of the number of data points  If are estimating the age of a star to that of 1Gyr but we have 100 stars, we might hope to resolve detail to 100Myrs.  At some point if we get some estimate for 1 cluster we could improve the no. data points by doing this on 10 clusters to get 100Myrs down to 10Myrs!  DON’T BE TOO OPTOMISTIC, there are of course other limiting factors so we couldn’t get down to 1Myr.  Prefis numbers with “in the best scenario”, “of course we won’t reach this because…”  Read up on neural nets and look up “sibenco’s universal function approximation theorem”  “look there is a universal approximation theorem that was proved 30 years ago so what we plan should work” – sounds good i.e. what we plan on doing will because of pre-existing knowledge  Add science and referencing |
| Co-marked by either:  Vechio loves hierarchical modelling but dislikes over optimism  Speake: all about referencing  Chaplin: all about the typos |
|  |

∝ μσθΦ

Did you specify it to do it for N parameters?

Next week: discuss reparametrizing  
 proposal template  
 try the problems