FIT2086 Studio 3
Introduction to Parameter Estimation

Daniel F. Schmidt

August 8, 2019

Contents

1 Introduction

Today's Studio covers several problems and questions regarding parameter estimation. We will first look at using maximum likelihood to derive an estimator for a simple problem. Then we will look at how to use estimated parameters to make predictions about future data. There is an additional question studying behaviour of the mean and median that should be completed out of studio time for those who are interested. You should complete the questions using either R, or by hand as appropriate.

2 ML Estimation of the Bernoulli Distribution

In this question we look at maximum likelihood estimate of the probability parameter θ in the Bernoulli distribution. Remember that the probability distribution for Bernoulli distribution with probability θ is:

$$\mathbb{P}(y \,|\, \theta) = \theta^y (1-\theta)^{1-y} \tag{1}$$

where $y \in \{0, 1\}$ is a binary variable. If Y comes from a Bernoulli distribution with probability θ, we can write $Y \sim \text{Be}(\theta)$. Let $\mathbf{y} = (y_1, \ldots, y_n)$ be a vector of n binary random variables (0 or 1). We can model this data using n independent and identical Bernoulli distributions (1). We are interested in trying to estimate the unknown θ from our observations $\mathbf{y}$ using maximum likelihood.

1. First, write down the likelihood of the sequence $\mathbf{y}$ if each y_i is modelled using an independent Bernoulli distribution with the same probability θ, i.e., what is:

$$p(\mathbf{y} \,|\, \theta) = \prod_{i=1}^{n} p(y_i \,|\, \theta)$$

   equal to? This likelihood would appropriate if we believed that the same coin was used in all n trials to generate our sequence.

2. Next, take the negative logarithm (using the natural logarithm ln()) of the likelihood to find the negative log-likelihood.

3. Now, given the negative log-likelihood, we want to find the maximum likelihood estimator of the probability θ for a given data sample $\mathbf{y}$, which we will call $\hat{\theta}_{\text{ML}}(\mathbf{y})$. To do this, (i) differentiate your negative log-likelihood function with respect to θ, (ii) set the derivative to zero and (iii) solve for θ. Write down the steps of your derivation.

4. Can you connect the resulting maximum likelihood estimator to any quantity you have seen before?

5. If you were going to use the ML estimate of θ to make predictions about the likelihood of seeing successes in future data, can you see any possible problems that might arise with any particular datasets $\mathbf{y}$ used to estimate θ?

3 Estimation and Simple Prediction

In this question we are going to use a Gaussian distribution to model some data, and use the fitted model to make some predictions about future data from the same source. The probability density function for a normal is

$$p(x \mid \mu, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right), \tag{2}$$

where μ is the mean, σ^2 is the variance, and σ is therefore the standard deviation.

The idea here is that we have some data and we propose a normal distribution as a possible model for the unknown population. We will then use the samples we have obtained from the population to try and guess the "best" values of the mean μ and variance σ^2 of our normal model. Once we have guessed good values for the parameters μ and σ^2, we can plug these into our normal distribution and use the resulting estimated ("fitted") model to make predictions about the population.

Note that we do not necessarily *believe* that our population is truly normally distributed – we are just using the normal distribution as a potential model to approximate our population. In this question, we will also be given more data from our population which we can use to assess how well our model actually captures the properties of the population. In reality we wouldn't usually have access to "future" data like this!

Load the file `heights.csv` into R; this contains two data frames – the first, `train`, contains the data we will use to fit ("train") our normal distribution. The second, `test`, contains additional data from the same source that we will use to assess our predictions on.

1. Write an R function that takes a vector of data, and calculates the maximum likelihood estimates of μ and σ^2:

   $$\hat{\mu}_{\mathrm{ML}}(\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} y_i, \quad \hat{\sigma}^2_{\mathrm{ML}}(\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\mu}_{\mathrm{ML}}(\mathbf{y}))^2.$$

   In addition, your function should also compute the unbiased estimate of the variance:

   $$\hat{\sigma}^2_{\mathrm{u}}(\mathbf{y}) = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \hat{\mu}_{\mathrm{ML}}(\mathbf{y}))^2.$$

   Remember, the unbiased estimator of variance is such that if we *repeatedly* drew different data samples from the population and calculated the variance using $\hat{\sigma}^2_{\mathrm{u}}(\mathbf{y})$ for each of these samples, then we would neither over- nor under-estimate the true population variance σ^2 *on average*.

   Your R function should return all three estimates using a list.

2. Using the function you have written, fit a normal distribution to the column of data in the dataframe `train`. How do the two estimates of variance (maximum likelihood vs unbiased) differ?

3. Recreate the figure in slide titled "Normal Example (3)" in Lecture 3 using your data and the two normal distributions you have fitted. *(Hints: (i) Start by plotting the data points along the x-axis, then use the* `lines()` *function to overlay your two normal probability density functions. (ii) You can use* `dnorm()` *when plotting the pdf. (iii) remember you can use the* `xlim` *and* `ylim` *option in the* `plot()` *function to set the x- and y-limits to appropriate values.)* What is the difference between the two estimated densities?

4. Finally, we can use our fitted models to make some predictions about the population from which our data came. We do this by "plugging" our estimates of μ and σ^2 into the normal PDF. This is often called a "plug-in" predictive density or distribution. We can then use the `pnorm()` function to calculate:

(a) The probability of someone from our population being taller than $1.7\,m$.

(b) The probability of someone from our population being shorter than $1.5\,m$.

(c) The probability of someone from our population being between $1.6m$ and $1.75m$ in height.

Do this using both estimates of variance (maximum likelihood and the unbiased estimate).

5. We can test how well our predictions performed by seeing how close the probabilities are to the empirical (observed) probabilities in our test data. Estimate these probabilities using a command like:

$$\texttt{mean(test\$height} > 1.7\texttt{)}$$

which will calculate the proportion of people in the testing data set that have a height greater than $1.7\,m$. This works by first checking each value of `height` and returning a `TRUE` if it is greater than 1.7, and a `FALSE` if it is less than or equal to 1.7 (try `head(test$height)` and `head(test$height > 1.7)` to see this). R interprets `TRUE` as a "1", and `FALSE` as a "0". Then the `mean()` calculates the proportion of times a `TRUE` appears, which tells how frequently (i.e., the probability) of a height in the test data being greater than 1.7. By the law of large numbers (see Lecture 2), when the test data is big enough this empirical proportion will be very close to the *true* population proportion.

Compare these to your predicted probabilities using the ML and unbiased estimates of variance. How accurate are the probabilities predicted by the two different estimates?

6. One very general way of measuring the predictive performance of a model is to calculate the negative log-likelihood of the data under the model. The better an estimate of the probabilities/frequencies of possible data values arising that a model provides, the higher the likelihood (and therefore, lower the negative log-likelihood) the model will achieve on new data.

The negative log-likelihood calculated on new data measures how well the probabilities predicted by our fitted model match the actual frequency of the different future data values occurring in the population. Remember that the negative-log of a small probability will be a large number, and the negative-log of a large probability will be a small number. Therefore, the better the model approximates the true population, the greater the probability it will assign to *typical* values of data that will arise in the population, and the smaller the negative-log-likelihood on the future data will be. In contrast, the worse the model approximates the population, the less probability it will assign to typical values of data in the population, and ergo, the greater the negative log-likelihood. The negative log-likelihood is therefore a good measure of how well the probabilities you predicted about the population match the actual probabilities of different data values in the population.

Write an R function `norm_negloglike(y,mu,sigma)` that takes a vector of data `y`, an mean `mu` and a variance `sigma` parameter, and returns the negative log-likelihood of the data under a normal model with $\mu =$ `mu` and $\sigma =$ `sigma`, i.e,

$$-\log p(\mathbf{y} \mid \mu, \sigma^2) = \frac{n}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2.$$

Use this to calculate the negative log-likelihood of the testing data in `test$heights` using the normal models with both the maximum likelihood estimate of variance and the unbiased estimate of variance. Which one seems to fit the population better by this metric?

4 Additional Question: Median vs Mean

This question can be completed out of Studio time for those who are interested. In this question we will use simulation to examine the behaviour of the sample median and sample mean as estimators of "centrality", i.e., the average value of a dataset. We know that the sample mean formula uses the value of every datapoint in the sample $\mathbf{y} = (y_1, \ldots, y_n)$, while the median uses the actual value of only (at most) the central two values; the values of the other samples are used only to establish which one (or two) samples are in the middle of the sorted list of samples.

The obvious question then is, which is a better estimator, and when is one better than the other? We can use the ideas presented in Lecture 3 on evaluating estimators using their sampling distribution to try and answer this question.

1. Write an R function `mean_median_test(niterations, mu, sigma, n)`. This function should contain a loop that repeatedly generates random datasets of size `n` from a normal distribution with mean `mu` and standard deviation `sigma`, and for each dataset, it should calculate both the sample mean and sample median and store these in vectors. This loop should repeat `niterations` times. Be aware in R your code can be much slower if you do not preallocate your arrays, so you should do this whenever you can. In this case, you could create an array to store the sample means and medians from each of the datasets using the `rep()` function, e.g., `mu_hat = rep(0, niterations)` will create an array of length `niterations` that is full of zeros.

   Once the loop has finished, your program should calculate: (i) the bias of both estimators; (ii) the variance of both estimators; (iii) the squared error of both estimators. These quantities should be calculated using `mu` (i.e., the mean parameter used to generate the data) as the "true" value you are comparing against; for example, the bias can be estimated using something like

   $$\texttt{retval\$bias} = \texttt{mean}(\texttt{mu\_hat} - \texttt{mu})$$

   These can easily be calculated using the vectors of sample means and sample medians you calculated in your loop. These values should be returned using a list. It should also calculate the *relative squared error* of the mean, relative to the median using:

   $$\text{RelMSE} = \frac{\text{MSE(sample mean)}}{\text{MSE(sample median)}}$$

   where $\text{MSE}(\cdot)$ denotes the squared error.

2. Run your function for `niterations = 10000`, `n = 10`, `mu = 0`, `sigma = 1` and `n = 10`. How do the different values of bias, variance and squared error for the two estimators compare? Are either estimators biased? Which one is on average closer to the true value `mu` as measured by squared error?

   From Lecture 3 we know the exact variance, bias and squared error of the sample mean if the data comes from a normal distribution. How close are the values calculated by simulation?

3. Run your function again `niterations = 10000`, `mu = 0`, `sigma = 10` and `n = 10`. How do the values change? Why does the relative-squared error not (substantially) change?

4. Run your function again `niterations = 10000`, `mu = 0`, `sigma = 10` and `n = 100`. How do the values change?

5. Modify your function to take an additional argument `nc`. The parameter `nc` should be the number of samples to *contaminate*. What this means is that within our main loop, after our sample of `n` datapoints from the $N(\texttt{mu}, \texttt{sigma}^2)$ distribution has been generated, we replace the first `nc`

datapoints with numbers generated from a normal distribution with mean `mu` and standard deviation 4 `sigma`.

These are considered contaminated data points as they have bigger variance than we would expect, and mimic the sort of errors that can occur if data is incorrectly recorded, or outliers exist.

6. Run your function again with `niterations = 10000`, `mu = 0`, `sigma = 1`, `n = 10`, and `nc = 1`. How do the two estimators compare in this case? You can try increasing the number of contaminated datapoints and see how the two estimators change in behaviour.

7. Additional question: Run your function with `niterations = 10000`, `mu = 0` and `sigma = 1` and `nc = 0` for `n` from 5 to 100 at steps of 5, and create two plots: (i) the variances of the mean and median, plotted on the same plot against the sample size; and (ii) the relative error RelMSE plotted against the sample size. How do the two plots differ? You can also run these plots again but with `nc` = 2.