Lecture 3: Statistical Models

Jing Ma, Statistics, TAMU

4 September, 2019

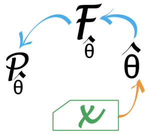
Recap

- Generative models for discrete data: Poisson, binomial, multinormial
- How to generate random observations from R: `rpois`, `rbinom`, `rmultinom`, etc.

This lecture

- Statistical modeling
- Goodness of fit
- Maximum likelihood estimation

Statistical modeling



In the previous lecture, the knowledge of both the generative model and the values of the parameters provided us with complete probabilistic models we could use for decision making – for instance, whether we had really found an epitope.

Statistical modeling

In many real situations, neither the generative model nor the parameters are known.

- We estimate them using the data we have collected.
- Statistical modeling works from the data *upwards* to a model that *might* plausibly explain the data. This upward–reasoning step is called statistical **inference**.

This lecture will discuss some distributions and estimation methods that serve as building blocks for inference.

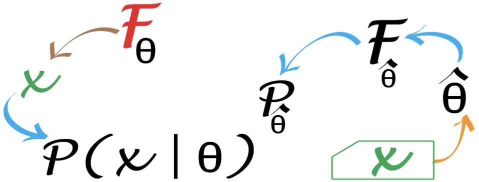
We *estimate* the parameters, denoted by Greek letters with what we call hats on them.

Parameters are the key

- Poisson(λ)
- Binomial ($\theta = (n, p)$)
- Multinomial ($\theta = (n, p_1, p_2, p_3, \ldots, p_{m-1})$)

The difference between statistical and probabilistic models

A probabilistic analysis is possible when we know a good generative model for the randomness in the data, *and* we are provided with the parameters' actual values.



$\mathcal{F}_\theta$
x
$\mathcal{P}(x \mid \theta)$
$\mathcal{P}_{\hat\theta}$
$\mathcal{F}_{\hat\theta}$
$\hat\theta$
x

Probabilistic models

- In the epitope example, knowing that false positives occurred as Bernoulli(0.01) per position, the number of patients assayed and the length of the protein ensured that there were *no unknown parameters*.

Probabilistic models

- In the epitope example, knowing that false positives occurred as Bernoulli(0.01) per position, the number of patients assayed and the length of the protein ensured that there were *no unknown parameters*.
- In such a case, we can use mathematical **deduction** to compute the probability of an event.

Probabilistic models

- In the epitope example, knowing that false positives occurred as Bernoulli(0.01) per position, the number of patients assayed and the length of the protein ensured that there were *no unknown parameters*.
- In such a case, we can use mathematical **deduction** to compute the probability of an event.
- In the epitope example, we used the Poisson probability as our **null model** with the given parameter $\lambda = 0.5$. We were able to conclude through mathematical deduction that the chances of seeing a maximum value of 7 or larger was smaller than 10^{-4} and thus that the observed data were highly unlikely under that model (i.e. under the null hypothesis).

Statistical modeling

- Now suppose that we know the number of patients and the length of the proteins (these are given by the experimental design) but not the distribution itself and the false positive rate.
- Once we observe data, we need to go *up* from the data to estimate both a probability model F (Poisson, normal, binomial) and eventually the missing parameter(s) for that model.

A simple example of statistical modeling

There are two parts of the modeling procedure.

- First we need a reasonable probability *distribution* to model the data generation process.
- Discrete count data may be modeled by simple probability distributions such as binomial, multinomial or Poisson distributions.

The normal distribution, or bell shaped curve, is often a good model for continuous measurements. Distributions can also be more complicated mixtures of these elementary ones (more on this in the lecture on Mixtures.)

Start with the data

Let's revisit the epitope example from the previous chapter, starting without the tricky outlier.

```r
load("data/e100.RData")
e99 = e100[-which.max(e100)]
```

Goodness-of-fit: visual evaluation

- Our first step is to find a fit from candidate distributions; this requires consulting graphical and quantitative goodness-of-fit plots.
- For discrete data, we can plot a barplot of frequencies (for continuous data, we would look at the histogram).

Goodness-of-fit: visual evaluation

```r
barplot(table(e99), space = 0.8, col = "chartreuse4")
```

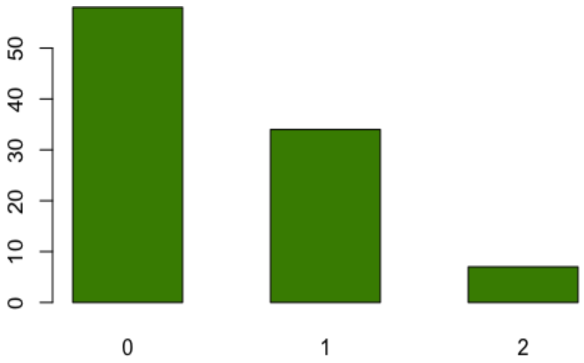


Fig. 1: The observed distribution of the epitope data without the outlier.

Goodness-of-fit: visual evaluation

- However, it is hard to decide which theoretical distribution fits the data best without using a comparison.

Goodness-of-fit: visual evaluation

- However, it is hard to decide which theoretical distribution fits the data best without using a comparison.
- One visual **goodness-of-fit** diagram is known as the **rootogram**: It hangs the bars with the observed counts from the theoretical red points. If the counts correspond exactly to their theoretical values, the bottom of the boxes will align exactly with the horizontal axis.

Goodness-of-fit: visual evaluation

```
gf1 = goodfit(e99, "poisson")
rootogram(gf1, xlab = "", rect_gp = gpar(fill = "chartreuse4"))
```

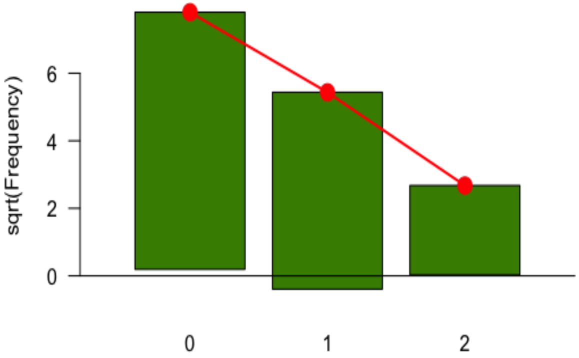
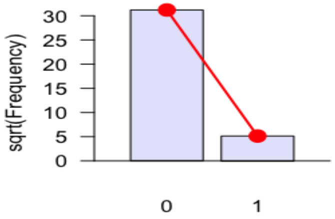


Fig. 2: Rootogram showing the square root of the theoretical values as red dots and the square root of the observed frequencies as drop down rectangles.

Goodness-of-fit: visual evaluation

To calibrate what such a plot looks like with a known Poisson variable, use `rpois` with $\lambda = 0.05$ to generate 100 Poisson distributed numbers and draw their rootogram.

```
simp = rpois(1000, lambda = 0.05)
gf2 = goodfit(simp, "poisson")
rootogram(gf2, xlab = "", rect_gp = gpar(fill = "lavender"))
```



Goodness-of-fit: visual evaluation

- We see that the rootogram for e99 seems to fit the Poisson model reasonably well. But remember, to make this happen we removed the outlier.
- The Poisson is completely determined by one parameter, often called the Poisson mean λ.
- We estimate the Poisson parameter from the data.

Parameter of the Poisson distribution

"The parameter is called the Poisson mean because it is the mean of the theoretical distribution *and*, as it turns out, is estimated by the sample mean."

Estimating the parameter of the Poisson distribution

- The most common way of estimating λ is to choose the value $\hat{\lambda}$ that makes the observed data the most likely.
- This is called the **maximum likelihood estimator**, often abbreviated as **MLE**.
- We will illustrate this idea later.

Good practice

- Although we took out the extreme observation before making a guess at the probability distribution, we are going to return to the data with it for the rest of our analysis.

Good practice

- Although we took out the extreme observation before making a guess at the probability distribution, we are going to return to the data with it for the rest of our analysis.
- In practice we would not know whether there is an outlier, and which data point(s) it is / they are.

Good practice

- Although we took out the extreme observation before making a guess at the probability distribution, we are going to return to the data with it for the rest of our analysis.
- In practice we would not know whether there is an outlier, and which data point(s) it is / they are.
- The effect of leaving it in is to make our estimate of the mean higher.

Good practice

- Although we took out the extreme observation before making a guess at the probability distribution, we are going to return to the data with it for the rest of our analysis.
- In practice we would not know whether there is an outlier, and which data point(s) it is / they are.
- The effect of leaving it in is to make our estimate of the mean higher.
- In turn, this would make it more likely that we'd observe a value of 7 under the null model, resulting in a larger p-value.

Good practice

- Although we took out the extreme observation before making a guess at the probability distribution, we are going to return to the data with it for the rest of our analysis.
- In practice we would not know whether there is an outlier, and which data point(s) it is / they are.
- The effect of leaving it in is to make our estimate of the mean higher.
- In turn, this would make it more likely that we'd observe a value of 7 under the null model, resulting in a larger p-value.
- So, if the resulting p-value is small even with the outlier included, we are assured that our analysis is up to something real.

Good practice

- Although we took out the extreme observation before making a guess at the probability distribution, we are going to return to the data with it for the rest of our analysis.
- In practice we would not know whether there is an outlier, and which data point(s) it is / they are.
- The effect of leaving it in is to make our estimate of the mean higher.
- In turn, this would make it more likely that we'd observe a value of 7 under the null model, resulting in a larger p-value.
- So, if the resulting p-value is small even with the outlier included, we are assured that our analysis is up to something real.
- We call such a tactic being **conservative**: we err on the side of caution, of not detecting something.

Estimating the parameter of the Poisson distribution

What value for the Poisson mean makes the data the most probable?

In the first step, we tally the outcomes.

```
table(e100)
```

```
## e100
##  0  1  2  7
## 58 34  7  1
```

Estimating the parameter of the Poisson distribution

Then we are going to try out different values for the Poisson mean and see which one gives the best fit to our data.

If the mean λ of the Poisson distribution were 3, the counts would look something like this:

```
table(rpois(100, 3))
```

```
##
##  0  1  2  3  4  5  6  7
##  8 19 22 16 16 10  8  1
```

This has many more 2's and 3's than we see in our data. So we see that $\lambda = 3$ is unlikely to have produced our data, as the counts do not match up so well.

Estimating the parameter of the Poisson distribution

So we could try out many possible values and proceed by brute force.

```
table(rpois(100, 2))
```

```
##
##  0  1  2  3  4  5  6  7
## 14 30 16 24  8  5  1  2
```

```
table(rpois(100, 1))
```

```
##
##  0  1  2  3  4  6
## 42 28 20  8  1  1
```

```
table(rpois(100, 0.75))
```

```
##
##  0  1  2  3
## 47 39 11  3
```

Likelihood of the Poisson distribution

- We can use a little mathematics to see which value maximizes the probability of observing our data.

$$\begin{aligned} & P(58 \times 0, 34 \times 1, 7 \times 2, \text{one } 7 \mid \text{data are Poisson}(m)) \\ = & P(0)^{58} \times P(1)^{34} \times P(2)^{7} \times P(7)^{1}. \end{aligned}$$

Likelihood of the Poisson distribution

- We can use a little mathematics to see which value maximizes the probability of observing our data.
- Let's calculate the probability of seeing the data if the value of the Poisson parameter is m.

$$\begin{aligned} &P(58 \times 0, 34 \times 1, 7 \times 2, \text{one } 7 \mid \text{data are Poisson}(m)) \\ = \ &P(0)^{58} \times P(1)^{34} \times P(2)^{7} \times P(7)^{1}. \end{aligned}$$

Likelihood of the Poisson distribution

- We can use a little mathematics to see which value maximizes the probability of observing our data.
- Let's calculate the probability of seeing the data if the value of the Poisson parameter is m.
- Since we suppose the data are from independent draws, this probability is simply the product of individual probabilities:

$$
\begin{aligned}
&P(58 \times 0, 34 \times 1, 7 \times 2, \text{one } 7 \mid \text{data are Poisson}(m)) \\
= &P(0)^{58} \times P(1)^{34} \times P(2)^{7} \times P(7)^{1}.
\end{aligned}
$$

Likelihood of the Poisson distribution

For $m = 3$ we can compute this

```
prod(dpois(c(0, 1, 2, 7), lambda = 3)^(c(58, 34, 7, 1)))
```

```
## [1] 1.392143e-110
```

Likelihood of the Poisson distribution

This probability is the **likelihood function** of λ, given the data, and we write it as

$$L(\lambda, \;\; x = (k_1, k_2, k_3, ...)) = \prod_{i=1}^{100} f(k_i).$$

- L stands for likelihood, and
- $f(k) = e^{-\lambda}\frac{\lambda^k}{k!}$ is the Poisson probability we saw earlier.

Instead of working with multiplications of a hundred small numbers, it is convenient to take the logarithm.

Likelihood of the Poisson distribution

We will compute the likelihood for many different values of the Poisson parameter.

To do this we need to write a small function that computes the probability of the data for different parameter values.

```
loglikelihood  =  function(lambda, data = e100) {
  sum(log(dpois(data, lambda)))
}
```

Likelihood of the Poisson distribution

Now we can compute the likelihood for a whole series of `lambda` values from 0.05 to 0.95.

```
lambdas = seq(0.05, 0.95, length = 100)
loglik = vapply(lambdas, loglikelihood, numeric(1))
plot(lambdas, loglik, type = "l", col = "red", ylab = "", lwd = 2,
     xlab = expression(lambda))
m0 = mean(e100);
abline(v = m0, col = "blue", lwd = 2)
abline(h = loglikelihood(m0), col = "purple", lwd = 2)
```

Likelihood of the Poisson distribution

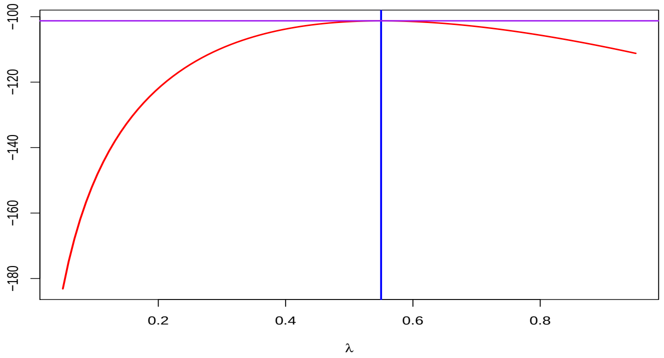


Fig. 3: The red curve is the log-likelihood function. The vertical line shows the value of ‘m‘ (the mean, 0.55) and the horizontal line the log-likelihood of ‘m‘. It looks like ‘m‘ maximizes the likelihood.

Model fit with `goodfit`

In fact there is a shortcut: the function `goodfit`.

```
gf  =  goodfit(e100, "poisson")
names(gf)
```

```
## [1] "observed" "count"    "fitted"   "type"     "method"   "df"
## [7] "par"
```

```
gf$par
```

```
## $lambda
## [1] 0.55
```

- The output of `goodfit` is a composite object called a list.
- One of its components is called `par` and contains the value(s) of the fitted parameter(s) for the distribution studied.
- In this case it's only one number, the estimate of λ.

Classical statistics for classical data

Here is a formal proof of our computation which found that the mean that maximizes the (log-)likelihood.

$$\log L(\lambda, x) = \sum_{i=1}^{100} -\lambda + k_i \log \lambda - \log(k_i!) \tag{1}$$

$$= -100\lambda + \log \lambda \left(\sum_{i=1}^{100} k_i \right) + \text{const.} \tag{2}$$

We use the catch-all "const." for terms that do not depend on λ (although they do depend on x, i.e., on the k_i).

Classical statistics for classical data

To find the λ that maximizes this, we compute the derivative in λ and set it to zero.

$$\frac{d}{d\lambda}\log L = -100 + \frac{1}{\lambda}\sum_{i=1}^{100} k_i \overset{?}{=} 0 \tag{3}$$

$$\lambda = \frac{1}{100}\sum_{i=1}^{100} k_i = \bar{k} \tag{4}$$

Classical statistics for classical data

- The first step of a *statistical approach* is to start ‘from the ground up’ (from the data) to infer the model parameter(s): this is called statistical **estimation** of a parameter from data.

Classical statistics for classical data

- The first step of a *statistical approach* is to start ‘from the ground up’ (from the data) to infer the model parameter(s): this is called statistical **estimation** of a parameter from data.
- Another important component will be choosing which family of distributions our data come from, which is done by evaluating the **goodness of fit**.

Classical statistics for classical data

- The first step of a *statistical approach* is to start 'from the ground up' (from the data) to infer the model parameter(s): this is called statistical **estimation** of a parameter from data.
- Another important component will be choosing which family of distributions our data come from, which is done by evaluating the **goodness of fit**.
- In the classical *statistical testing* framework, we consider one single model, which we call the *null model*, for the data.

Classical statistics for classical data

- The first step of a *statistical approach* is to start ‘from the ground up’ (from the data) to infer the model parameter(s): this is called statistical **estimation** of a parameter from data.
- Another important component will be choosing which family of distributions our data come from, which is done by evaluating the **goodness of fit**.
- In the classical *statistical testing* framework, we consider one single model, which we call the *null model*, for the data.
- The null model formulates an “uninteresting” baseline.

Classical statistics for classical data

- The first step of a *statistical approach* is to start 'from the ground up' (from the data) to infer the model parameter(s): this is called statistical **estimation** of a parameter from data.
- Another important component will be choosing which family of distributions our data come from, which is done by evaluating the **goodness of fit**.
- In the classical *statistical testing* framework, we consider one single model, which we call the *null model*, for the data.
- The null model formulates an "uninteresting" baseline.
- We then test whether there is something more interesting going on by computing the probability that the data are compatible with that model.

Models are concise but expressive representations of the data generating process.

- For the Poisson, for instance, knowing one number allows us to know everything about the distribution, including, as we saw earlier, the probabilities of extreme or rare events.

Binomial distributions and maximum likelihood

In a binomial distribution there are two parameters:

- the number of trials n, which is typically known, and
- the probability p of seeing a 1 in a trial. This probability is often unknown.

An example

Suppose we take a sample of $n = 120$ males and test them for red-green color blindness. We can code the data as 0 if the subject is not color blind and 1 if he is. We summarize the data by the table:

```r
cb  =  c(rep(0, 110), rep(1, 10))
table(cb)
```

```
## cb
##   0   1
## 110  10
```

Which value of p is the most likely given these data?

An example

Suppose we take a sample of $n = 120$ males and test them for red-green color blindness. We can code the data as 0 if the subject is not color blind and 1 if he is. We summarize the data by the table:

```
cb  =  c(rep(0, 110), rep(1, 10))
table(cb)
```

```
## cb
##   0   1
## 110  10
```

Which value of p is the most likely given these data? $\hat{p} = \frac{1}{12}$.

The MLE

```
probs  =  seq(0, 0.3, by = 0.005)
likelihood = dbinom(sum(cb), prob = probs, size = length(cb))
probs[which.max(likelihood)]
```

```
## [1] 0.085
```

- However, be careful: sometimes ML estimates are harder to guess and to compute, as well as being much less intuitive.

The MLE

```r
probs  =  seq(0, 0.3, by = 0.005)
likelihood = dbinom(sum(cb), prob = probs, size = length(cb))
probs[which.max(likelihood)]
```

```
## [1] 0.085
```

- However, be careful: sometimes ML estimates are harder to guess and to compute, as well as being much less intuitive.
- In this special case, your intuition may give you the estimate $\hat{p} = \frac{1}{12}$, which turns out to be the maximum likelihood estimate.

The MLE

```r
probs  =  seq(0, 0.3, by = 0.005)
likelihood = dbinom(sum(cb), prob = probs, size = length(cb))
probs[which.max(likelihood)]
```

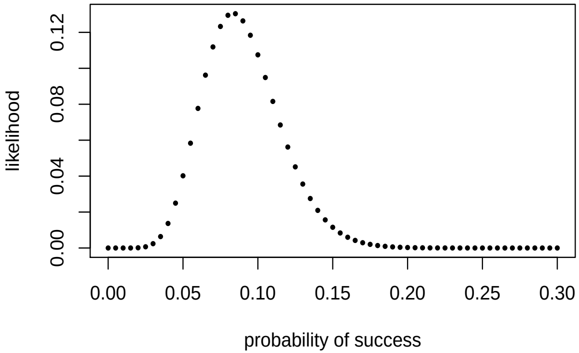
```
## [1] 0.085
```

- However, be careful: sometimes ML estimates are harder to guess and to compute, as well as being much less intuitive.
- In this special case, your intuition may give you the estimate $\hat{p} = \frac{1}{12}$, which turns out to be the maximum likelihood estimate.
- We put a hat over the letter to remind us that this is not (necessarily) the underlying true value, but an estimate we make from the data.

Plot of the likelihood as a function of the probabilities

The likelihood is a function on $[0, 1]$; here we have zoomed in, as the likelihood is practically zero for larger values of p.

```r
plot(probs, likelihood, pch = 16, xlab = "probability of success",
     ylab = "likelihood", cex=0.6)
```



The MLE

One can come up with different criteria than ML, which lead to other estimators: they all carry hats. We'll see other examples in the lecture on Mixtures.

Probability and likelihood

The probability and the likelihood are the same mathematical function,interpreted in different ways.

- In one case, it tells us how probable it is to see a particular set of values of the data, given the parameters.
- In the other case, we consider the data as fixed, and ask for the particular parameter value that makes the data more likely.

Likelihood for the binomial distribution

Suppose $n = 300$, and we observe $y = 40$ successes. Then, for the binomial distribution:

$$f(\theta \mid n, y) = f(y \mid n, \theta) = \binom{n}{y} \theta^y (1 - \theta)^{(n-y)}. \tag{5}$$

As $\binom{n}{y}$ is very large, we use the logarithm of the likelihood to give:

$$\log f(\theta|y) = 115 + 40 \log(\theta) + (300 - 40) \log(1 - \theta).$$

Likelihood for the binomial distribution

Here's a function we use to calculate it:

```r
loglikelihood = function(theta, n = 300, k = 40) {
  115 + k * log(theta) + (n - k) * log(1 - theta)
}
```

which we plot for θ ranging from 0 to 1.

Likelihood for the binomial distribution

```
thetas = seq(0, 1, by = 0.001)
plot(thetas, loglikelihood(thetas), xlab = expression(theta),
  ylab = expression(paste("log f(", theta, " | y)")),type = "l", ylim=c(-500,0)
```

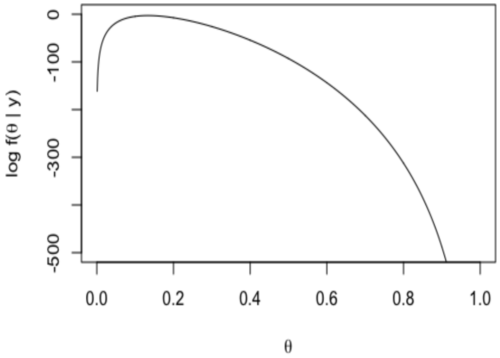


Fig. 4: The maximum lies at $40/300 = 0.1333...$, consistent with intuition, but we see that other values of theta are almost equally likely.

DNA count modeling

We use data from one strand of DNA for the genes of Staphylococcus aureus bacterium.

```
staph = readDNAStringSet("data/staphsequence.ffn.txt", "fasta")
staph[1] # the first gene
```

```
##   A DNAStringSet instance of length 1
##     width seq                                               names
## [1]  1362 ATGTCGGAAAAAGAAATTTGG...AAGAAATAAGAAATGTATAA lcl|NC_002952.2_c...
```

```
letterFrequency(staph[[1]], letters = "ACGT", OR = 0)
```

```
##   A   C   G   T
## 522 219 229 392
```

DNA count modeling

We want to know whether the first ten genes from these data come from the same multinomial.

```r
letterFrq = vapply(staph, letterFrequency, FUN.VALUE = numeric(4),
         letters = "ACGT", OR = 0)
colnames(letterFrq) = paste0("gene", seq(along = staph))
tab10 = letterFrq[, 1:10]
computeProportions = function(x) { x/sum(x) }
prop10 = apply(tab10, 2, computeProportions)
round(prop10, digits = 2)
```

```
##   gene1 gene2 gene3 gene4 gene5 gene6 gene7 gene8 gene9 gene10
## A  0.38  0.36  0.35  0.37  0.35  0.33  0.33  0.34  0.38   0.27
## C  0.16  0.16  0.13  0.15  0.15  0.15  0.16  0.16  0.14   0.16
## G  0.17  0.17  0.23  0.19  0.22  0.22  0.20  0.21  0.20   0.20
## T  0.29  0.31  0.30  0.29  0.27  0.30  0.30  0.29  0.28   0.36
```

DNA count modeling

- The ten vectors of probabilities vary a bit.
- Suppose the row mean p0 is the vector of multinomial probabilities for all the ten genes.

```
p0 = rowMeans(prop10)
p0
```

```
##         A         C         G         T
## 0.3470531 0.1518313 0.2011442 0.2999714
```

- We then use a Monte Carlo simulation to test whether the departures between the observed letter frequencies and expected values under this supposition are within a plausible range.

DNA count modeling

- First find out sums of nucleotide counts cs from each of the 10 genes (columns).

```
cs = colSums(tab10)
cs
```

```
##  gene1  gene2  gene3  gene4  gene5  gene6  gene7  gene8  gene9 gene10
##   1362   1134    246   1113   1932   2661    831   1515   1287    696
```

DNA count modeling

- Compute the expected counts by taking the outer product of p0 and cs.

```
expectedtab10 = outer(p0, cs, FUN = "*")
round(expectedtab10)
```

```
##   gene1 gene2 gene3 gene4 gene5 gene6 gene7 gene8 gene9 gene10
## A   473   394    85   386   671   924   288   526   447    242
## C   207   172    37   169   293   404   126   230   195    106
## G   274   228    49   224   389   535   167   305   259    140
## T   409   340    74   334   580   798   249   454   386    209
```

DNA count modeling

- Generate a random table with the correct column sums using the `rmultinom` function.

```
randomtab10 = sapply(cs, function(s) { rmultinom(1, s, p0) } )
all(colSums(randomtab10) == cs)
```

```
## [1] TRUE
```

DNA count modeling

- Evaluate the test statistics for 1000 replications $\rightarrow$ which are used to construct the empirical null distribution.
- S1 is the test statistic from the actual data.

```
stat = function(obsvd, exptd = 20 * pvec) {
   sum((obsvd - exptd)^2 / exptd)
}
B = 1000
simulstat = replicate(B, {
  randomtab10 = sapply(cs, function(s) { rmultinom(1, s, p0) })
  stat(randomtab10, expectedtab10)
})
S1 = stat(tab10, expectedtab10)
sum(simulstat >= S1)
```

```
## [1] 0
```

DNA count modeling

```r
hist(simulstat, col = "lavender", breaks = seq(0, 75, length.out=50))
abline(v = S1, col = "red")
abline(v = quantile(simulstat, probs = c(0.95, 0.99)),
       col = c("darkgreen", "blue"), lty = 2)
```

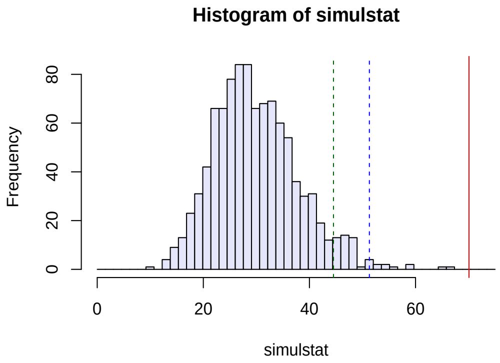
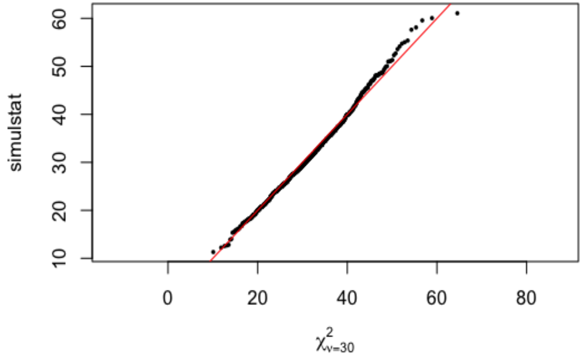


Fig. 5: Histogram of ‘simulstat‘. The value of ‘S1‘ is marked by the vertical red line, those of the 0.95 and 0.99 quantiles by the dashed lines.

DNA count modeling: the QQ-plot

```
qqplot(qchisq(ppoints(2000), df = 30), simulstat, main = "",
  xlab = expression(chi[nu==30]^2), asp = 1, cex = 0.5, pch = 16)
abline(a = 0, b = 1, col = "red")
```



Our simulated statistic's distribution compared to χ^2_{30} using a QQ-plot, which shows the theoretical quantiles for the χ^2_{30} distribution on the horizontal axis and the sampled ones on the vertical axis.

Chargaff's Rule

The most important pattern in the nucleotide frequencies was discovered by Chargaff.

Before DNA sequencing was available, using the weight of the molecules, he asked whether the nucleotides occurred at equal frequencies. He called this the tetranucleotide hypothesis: $p_A = p_C = p_G = p_T$.

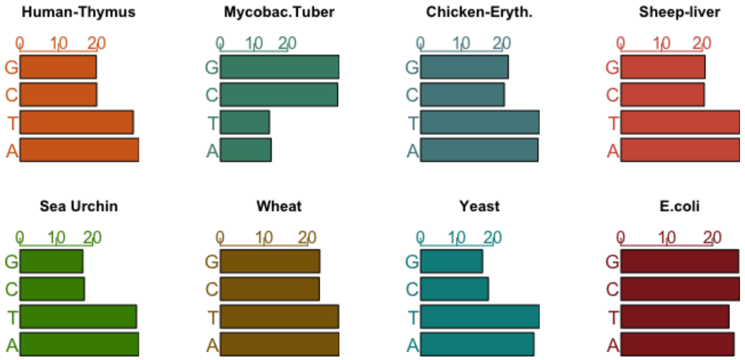
Chargaff's Rule

Unfortunately, Chargaff only published the *percentages* of the mass present in different organisms for each of the nucleotides, not the measurements themselves.

```
load("data/ChargaffTable.RData")
ChargaffTable
```

```
##                   A    T    C    G
## Human-Thymus   30.9 29.4 19.9 19.8
## Mycobac.Tuber  15.1 14.6 34.9 35.4
## Chicken-Eryth. 28.8 29.2 20.5 21.5
## Sheep-liver    29.3 29.3 20.5 20.7
## Sea Urchin     32.8 32.1 17.7 17.3
## Wheat          27.3 27.1 22.7 22.8
## Yeast          31.3 32.9 18.7 17.1
## E.coli         24.7 23.6 26.0 25.7
```

Chargaff's Rule



- Do these data seem to come from equally likely multinomial categories?
- Can you suggest an alternative pattern?
- Can you do a quantitative analysis of the pattern, perhaps inspired by the simulations above?

Base pairing

- Chargaff postulated a pattern called *base pairing*, which ensured a perfect match of the amount of adenine (A) in the DNA of an organism to the amount of thymine (T).
- Similarly, whatever the amount of guanine (G), the amount of cytosine (C) would be the same.
- This is now called Chargaff's rule.
- On the other hand, the amount of C/G in an organism could be quite different from that of A/T, with no obvious pattern across organisms.

Base pairing

Based on Chargaff's rule, we might define a statistic

$$(p_{\mathsf{C}} - p_{\mathsf{G}})^2 + (p_{\mathsf{A}} - p_{\mathsf{T}})^2,$$

summed over all rows of the table.

We are going to look at a comparison between the data and what would occur if the nucleotides were **exchangeable**, in the sense that the probabilities observed in each row were in no particular order, so that there were no special relationship between the proportions of As and Ts, or between those of Cs and Gs.

Base pairing

Write a function to calculate the empirical null distribution.

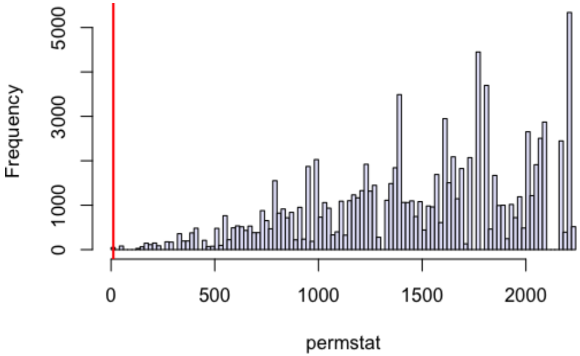
```
statChf = function(x){
  sum((x[, "C"] - x[, "G"])^2 + (x[, "A"] - x[, "T"])^2)
}
chfstat = statChf(ChargaffTable)
permstat = replicate(100000, {
     permuted = t(apply(ChargaffTable, 1, sample))
     colnames(permuted) = colnames(ChargaffTable)
     statChf(permuted)
})
pChf = mean(permstat <= chfstat)
pChf
```

```
## [1] 0.00013
```

The null distribution and the test statistic

- Histogram of our statistic `statChf` computed from simulations using per-row permutations of the columns.
- The value it yields for the observed data is shown by the red line.

```r
hist(permstat, breaks = 100, main = "", col = "lavender")
abline(v = chfstat, lwd = 2, col = "red")
```



What does it mean?

- The histogram shows that it is quite rare to have a value as small as the observed 11.1, where the red line is drawn. The probability of observing a value as small or smaller is $\texttt{pChf}=1.3 \times 10^{-4}$.
- Thus the data strongly support Chargaff's conjecture.

Question

- When computing `pChf`, we only looked at the values in the null distribution smaller than the observed value. Why did we do this in a one-sided way here?

A special multinomial: Hardy-Weinberg equilibrium

- Suppose the overall frequency of allele M in the population is p, so that of N is $q = 1 - p$.
- The **Hardy-Weinberg equilibrium** (HWE) happens if there is independence of the frequencies of M and N.
- This would be the case if there is random mating in a large population with equal distribution of the alleles among sexes. The probabilities of the three genotypes are then as follows:

$$p_{\mathrm{MM}} = p^2, \quad p_{\mathrm{NN}} = q^2, \quad p_{\mathrm{MN}} = 2pq$$

A special multinomial: Hardy-Weinberg equilibrium

- We only observe the frequencies $(n_{\text{MM}},\ n_{\text{MN}},\ n_{\text{NN}})$ for the genotypes MM, MN, NN and the total number $S = n_{\text{MM}} + n_{\text{MN}} + n_{\text{NN}}$.
- We can write the likelihood, i.e., the probability of the observed data when the probabilities of the categories are given by $(p^2, q^2, 2pq)$, using the multinomial formula

$$
\begin{aligned}
&P(n_{\text{MM}},\ n_{\text{MN}},\ n_{\text{NN}} \mid p) \\
=& \binom{S}{n_{\text{MM}}, n_{\text{MN}}, n_{\text{NN}}} (p^2)^{n_{\text{MM}}} \times (2pq)^{n_{\text{MN}}} \times (q^2)^{n_{\text{NN}}}.
\end{aligned}
$$

Maximum likelihood estimation of HWE

The log-likelihood under HWE is

$$L(p) = n_{\mathrm{MM}} \log(p^2) + n_{\mathrm{MN}} \log(2pq) + n_{\mathrm{NN}} \log(q^2).$$

The value of p that maximizes the loglikelihood is

$$p = \frac{n_{\mathrm{MM}} + n_{\mathrm{MN}}/2}{S}.$$

Given the data $(n_{\mathrm{MM}}, n_{\mathrm{MN}}, n_{\mathrm{NN}})$, the log-likelihood L is a function of only one parameter, p.

A data example: the log-likelihood

```r
library("HardyWeinberg")
data("Mourant")
Mourant[214:216,]
```

```
##     Population    Country Total  MM  MN  NN
## 214    Oceania Micronesia   962 228 436 298
## 215    Oceania Micronesia   678  36 229 413
## 216    Oceania     Tahiti   580 188 296  96
```

```r
nMM = Mourant$MM[216]
nMN = Mourant$MN[216]
nNN = Mourant$NN[216]
loglik = function(p, q = 1 - p) {
  2 * nMM * log(p) + nMN * log(2*p*q) + 2 * nNN * log(q)
}
xv = seq(0.01, 0.99, by = 0.01)
yv = loglik(xv)
```

A data example: the log-likelihood

```r
plot(x = xv, y = yv, type = "l", lwd = 2,
     xlab = "p", ylab = "log-likelihood")
imax = which.max(yv)
abline(v = xv[imax], h = yv[imax], lwd = 1.5, col = "blue")
abline(h = yv[imax], lwd = 1.5, col = "purple")
```

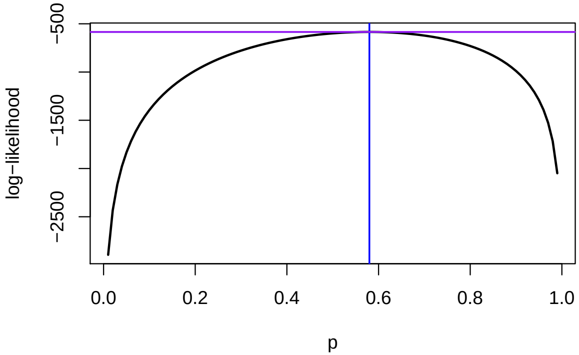


Fig. 6: Plot of the log-likelihood for the 216th observation.

A data example: the MLE

We can compute $\hat{p}_{MM}$, $\hat{p}_{MN}$ and $\hat{p}_{NN}$ using the `af` function from the **HardyWeinberg** package.

```
phat  =  af(c(nMM, nMN, nNN))
phat
```

```
## [1] 0.5793103
```

```
pMM   =  phat^2
qhat  =  1 - phat
```

The expected values under Hardy-Weinberg equilibrium are then

```
pHW = c(MM = phat^2, MN = 2*phat*qhat, NN = qhat^2)
sum(c(nMM, nMN, nNN)) * pHW
```

```
##       MM       MN       NN
## 194.6483 282.7034 102.6483
```

Visual comparison to the Hardy-Weinberg equilibrium

- A visual evaluation of the goodness-of-fit of Hardy-Weinberg was designed by de Finetti
- It places every sample at a point whose coordinates are given by the proportions of the alleles.
- We use the `HWTernaryPlot` functon to display the data and compare it to Hardy-Weinberg equilibrium graphically.

```r
par(mai = rep(0.1, 4))
pops = c(1, 69, 128, 148, 192)
genotypeFrequencies = as.matrix(Mourant[, c("MM", "MN", "NN")])
HWTernaryPlot(genotypeFrequencies[pops, ], markerlab = Mourant$Country[pops],
              alpha = 0.0001, curvecols = c("red", rep("purple", 4)),
              mcex = 0.75, vertex.cex = 1)
```

The de Finetti plot

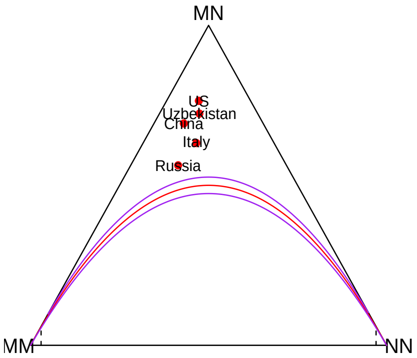


Fig. 7: This de Finetti plot shows the points as barycenters of the three genotypes using the frequencies as weights on each of the corners of the triangle. The Hardy-Weinberg model is the red curve, the acceptance region is between the two purple lines. We see that the US is the furthest from being in HW equilibrium.

Sequence motifs and logos

- The Kozak Motif occurs close to the start codon **ATG** of a coding region.
- The start codon itself always has a fixed spelling but in the five positions to the left of it, there is a nucleotide pattern in which the letters are quite far from being equally likely.
- We summarize this by giving the **position weight matrix** (PWM) or **position-specific scoring matrix** (PSSM), which provides the multinomial probabilities at every position. This is encoded graphically by the **sequence logo** package.

Sequence motifs and logos

```r
library("seqLogo")
load("data/kozak.RData")
kozak
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## A 0.33 0.25  0.4 0.15 0.20    1    0    0 0.05
## C 0.12 0.25  0.1 0.40 0.40    0    0    0 0.05
## G 0.33 0.25  0.4 0.20 0.25    0    0    1 0.90
## T 0.22 0.25  0.1 0.25 0.15    0    1    0 0.00
```

Sequence motifs and logos

```
pwm = makePWM(kozak)
seqLogo(pwm, ic.scale = FALSE)
```

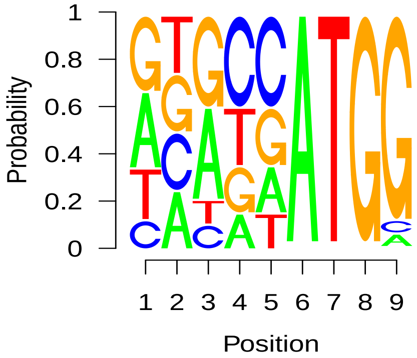


Fig. 8: A diagram called a sequence logo for the position dependent multinomial used to model the Kozak motif. It codifies the amount of variation in each of the positions on a log scale. The large letters represent positions where there is no uncertainty about which nucleotide occurs.

The Biostrings package

```r
library("Biostrings")
```

- The **Biostrings** package provides tools for working with sequence data.
- The essential data structures, or *classes* in **R**, are *DNAString* and *DNAStringSet*.
- These enable us to work with one or multiple DNA sequences efficiently.

Run the code yourself

```r
GENETIC_CODE
IUPAC_CODE_MAP
vignette(package = "Biostrings")
vignette("BiostringsQuickOverview", package = "Biostrings")
```

- The first line prints genetic code information.
- The second one returns IUPAC nucleotide ambiguity codes.
- The third line lists all the vignettes available in the **Biostrings** package.
- The fourth displays one particular vignette.

The BSgenome package

The **BSgenome** package provides access to many genomes, and there are also data packages that contain the whole genome sequences.

```r
library("BSgenome")
ag = available.genomes()
length(ag)
```

```
## [1] 91
```

```r
ag[1:2]
```

```
## [1] "BSgenome.Alyrata.JGI.v1"
## [2] "BSgenome.Amellifera.BeeBase.assembly4"
```

Example: occurrence of a nucleotide pattern in a genome

- We are going to explore the occurrence of the AGGAGGT motif in the geonome of E.coli.
- We use the genome sequence of one particular strain, **Escherichia coli** str K12 substr.DH10B, whose NCBI accession number is NC_010473.

```
library("BSgenome.Ecoli.NCBI.20080805")
Ecoli
shineDalgarno = "AGGAGGT"
ecoli = Ecoli$NC_010473
```

Example: occurrence of a nucleotide pattern in a genome

We can count the pattern's occurrence in windows of width 50000 using the `countPattern` function.

```r
window = 50000
starts = seq(1, length(ecoli) - window, by = window)
ends   = starts + window - 1
numMatches = vapply(seq_along(starts), function(i) {
  countPattern(shineDalgarno, ecoli[starts[i]:ends[i]],
               max.mismatch = 0)
  }, numeric(1))
table(numMatches)
```

```
## numMatches
##  0  1  2  3  4
## 48 32  8  3  2
```

Example: occurrence of a nucleotide pattern in a genome

Poisson distribution is a good candidate, as a quantitative and graphical evaluation for these data shows.

```r
library("vcd")
gf = goodfit(numMatches, "poisson")
summary(gf)
```

```
##
## 	 Goodness-of-fit test for poisson distribution
##
##                       X^2 df  P(> X^2)
## Likelihood Ratio 4.134932  3 0.2472577
```

Example: occurrence of a nucleotide pattern in a genome

```r
distplot(numMatches, type = "poisson")
```

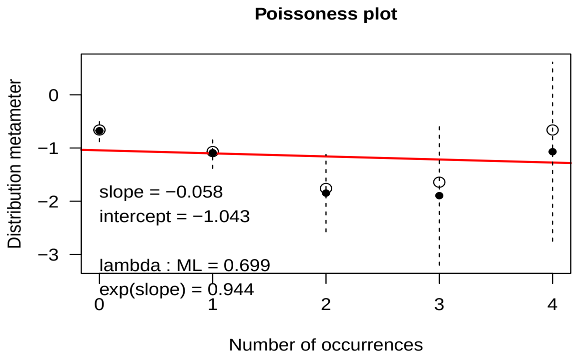


Fig. 9: Evaluation of a Poisson model for motif counts along the sequence: the open points show the observed count metameters; the filled points show the confidence interval centers; the dashed lines show the confidence intervals for each point.

Gap size between motifs

We can inspect the matches using the `matchPattern` function.

```
sdMatches = matchPattern(shineDalgarno, ecoli, max.mismatch = 0)
```

The `sdMatches` object contains the locations of all 65 pattern matches, represented as a set of so-called *views* on the original sequence. Now what are the distances between them?

```
betweenmotifs = gaps(sdMatches)
# width(betweenmotifs)
```

Model for the gap sizes

- If the motifs occur at random locations, we expect the gap lengths to follow an exponential distribution.
- The code below assesses this assumption. If the exponential distribution is a good fit, the points should lie roughly on a straight line.
- The exponential distribution has one parameter, the rate, and the line with slope corresponding to an estimate from the data is also shown.

Exponential for the gap sizes

```
library("Renext")
expplot(width(betweenmotifs), rate = 1/mean(width(betweenmotifs)),
        labels = "fitted exponential")
```

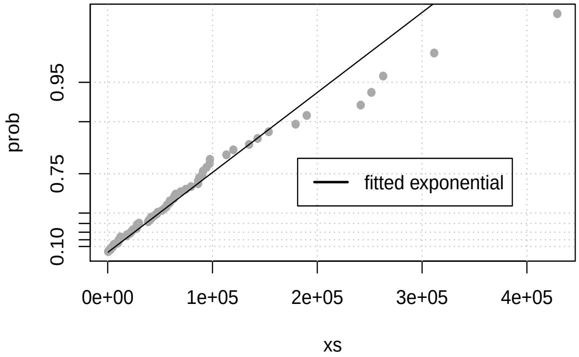


Fig. 10: There appears to be a slight deviation from the fitted line in the figure at the right tail of the distibution, i.e., for the large values.

Summary of this lecture

- Goodness of fit
- Mximum likelihood estimation