Lecture 14: Information theory

```
install.packages(c('devtools','coda','mvtnorm','loo'))
library(devtools)
install_github("rmcelreath/rethinking")
```

```
library(rstan)
library(shinystan)
library(car)
library(mvtnorm)
library(rethinking)
library(MASS)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
source("../utilityFunctions.R")
```

** This lecture is based on chapter 6 of Statistical Rethinking by Richard McElreath.*

Now we have learned how to create regressions in a Bayesian context, include multiple predictors and linear interactions between them, and interpret the results. But just because we *can* add predictors and interactions doesn't necessarily mean we *should*.

Good models—Bayesian or not—strike an optimal balance between

1. *Underfitting: poor prediction by learning too little from the data*
    - Also called bias—the difference between an estimator's expected value and the "true value"
2. *Overfitting: poor prediction by learning too much from the data*
    - sometimes called variance—error from sensitivity to small fluctuations in the data. Essentially, modeling noise rather than process.

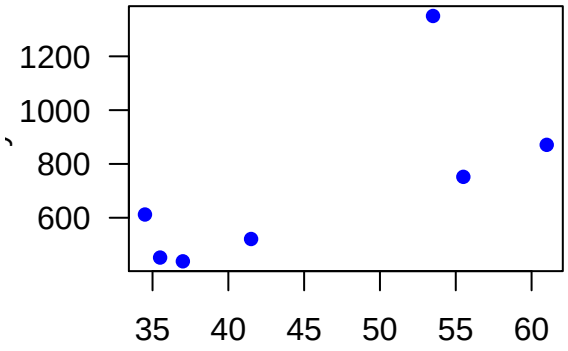
Adding variables without thought has two problems:

The first is that adding variables always improves the fit of the model—at least in the context of the data used to fit the model.

- Consider R^2, often interpreted as the "variance explained".
    - R^2 increases with predictor variables regardless of whether they matter.

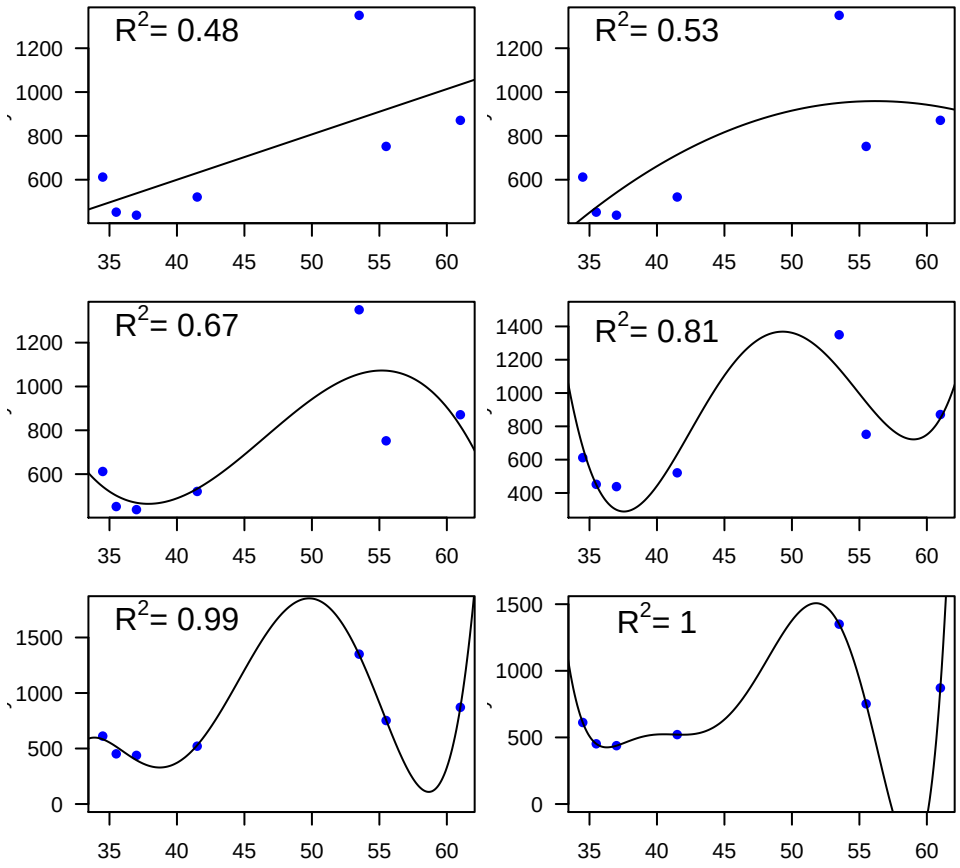
As an example, consider the following data:

```
x <- c(37, 35.5, 34.5, 41.5, 55.5, 61, 53.5)
y <- c(438, 452, 612, 521, 752, 871, 1350)
par(mar=c(3,3.2,0.1,0.5))
plot(x,y, pch=16, col="blue", las=1)
```



As a first step, we might model this in an OLS framework as a straight line. But why limit ourselves to a straight line? We could fit a quadratic, or cubic, or quintic model with polynomials.

```
m1 <- lm(y ~ x) # straight line
m2 <- lm(y ~ x + I(x^2)) # quadratic
m3 <- lm(y ~ x + I(x^2) + I(x^3)) # cubic
m4 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4)) # 4th degree
m5 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5)) # 5th degree
m6 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6)) # 6th degree
```



As we add polynomial terms, the fit improves. The 5th order model has an $R^2 = 0.99$, and the 6th order model passes through every point with no residual variance.

Another way to think about overfitting models is to consider model construction as data compression:

1. Parameters summarize relationships among the data by "compressing" the data into a simpler form.

- Compression is "lossy" because information is lost.

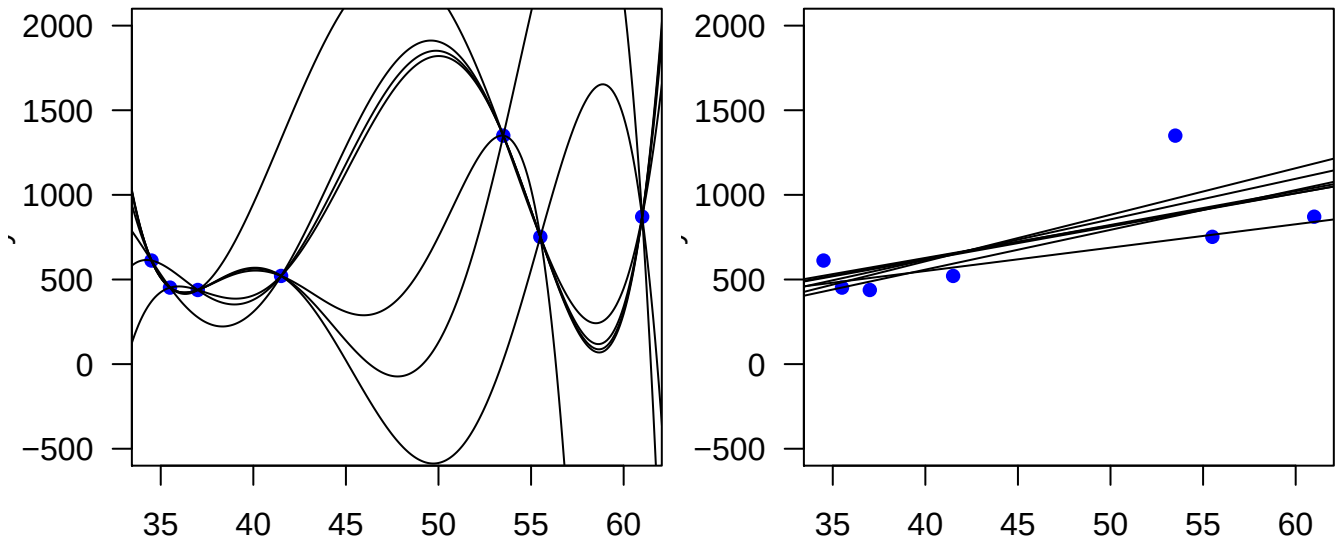
2. The parameters can then be used to generate new data, effectively decompressing the data.

- Our 6th-order polynomial model has a parameter for each data point. there is no compression! We just encode the data using parameters instead and learn nothing new.

The second problem with more complex models is that they fit the data better but often predict new data worse.

- Consider what happens when we drop out each data point in our last example one at a time and then fit 5th vs. 1st (i.e., straight line) order polynomial models (NOTE: this is cross-validation):

```
par(mar=c(3,3.2,0.1,0.5))
par(mfrow=c(1,2))
plot(x,y, pch=16, col="blue", ylim=c(-500,2000), las=1)
for(i in 1:length(x)){
  x1 <- x[-i]; y1 <- y[-i]
  m <- lm(y1 ~ x1 + I(x1^2) + I(x1^3) + I(x1^4) + I(x1^5))
  xz <- seq(20,65,0.1)
  yz <- predict(m, new=data.frame(x1=xz))
  lines(xz, yz)
}

plot(x,y, pch=16, col="blue", ylim=c(-500,2000), las=1)
for(i in 1:length(x)){
  x1 <- x[-i]; y1 <- y[-i]
  m <- lm(y1 ~ x1)
  xz <- seq(20,65,0.1)
  yz <- predict(m, new=data.frame(x1=xz))
  lines(xz, yz)
}
```



The polynomial curves are much more sensitive to the composition of the sample and thus fluctuate considerably depending on the exact sample composition.

- The simpler "underfit" model is insensitive, changing little as points are dropped.

This is a general contrast between overfitting and underfitting models: sensitivity to sample composition vs. potentially failing to describe the regular features of the sample.

There are two major (and non-mutually exclusive) ways that we can strike the balance between underfitting and overfitting

1. *Regularizing priors* keep the model from getting too excited about the data. This is a Bayesian version of ridge regression/lasso/elastic nets.
   - Shrink parameter estimates towards zero unless there is considerable information.
2. *Infomation criteria* for model selection and/or model averaging.

NOTE: These are not substitutes for using your noggin!

Information theory

**** see Burnham and Anderson, 2002 and Harte's 2011 book on Maximum Entropy and Ecology for a full treatment of all of this*

Whether we use regularization and/or information criteria, we need a benchmark of model performance.

We will call this the *target*, and in our case the target is out-of-sample *deviance*.

There are two major aspects of defining our target:

1. *Cost-benefit analysis:* How much does it cost when we are wrong? How much do we win if we are correct?

Suppose you are predicting the weather to decide whether to carry an umbrella or not.

- The cost of predicting sun incorrectly means you may get soaked, fry your phone, get hypothermia, and die.
- Predicting rain incorrectly may mean you are encumbered by carrying an umbrella around unnecessarily.
  - The former has a much higher cost

2. *Accuracy in context:* Even if we ignore costs & benefits, we need to judge accuracy somehow that accounts for how much the model could possibly improve prediction.

Enter Information Theory where the basic insight is to ask: *How much is our uncertainty reduced by learning an outcome?*

Weather forcasts are uncertain until the actual weather occurs. The reduction in uncertainty is then how much "information" we glean from learning the outcome.

- We just need a precise definition of uncertainty so we have baseline measure of how hard it is to predict and how much improvement is possible. This is the definition of *information*: the reduction in uncertainty derived from learning the outcome. For this definition to be useful, we need a way to quantify the uncertainty intrinsic to a probability distribution.

If there are two possible weather outcomes on a particular day, rain or sun, then each event occurs with some probability, and the probabilities sum to one.

There are a myriad of ways to measure uncertainty, but the metric should:

1. Be continuous. Otherwise a small change in outcome probability could result in a huge increase in uncertainty;
2. Increase as the number of possible events increases;
3. Be additive. The uncertainty over all events should be the sum of the separate uncertainties.

There is only one function that satisfies these properties. If there are n different possibilities and event i has probability p_i, then the unique measure of uncertainty for the list of probabilities P is

$$H(P) = -\mathrm{E}[\log p_i] = -\sum_{i=1}^{n} p_i \log p_i.$$

This should look familiar to every ecologist in the room.

Note that this equation blows up when $p_i = 0$ because $\log 0 = -\infty$. Unacceptable! Fortunately, we can use L'H^{o}pital's rule: $\lim_{p_i \to 0} p_i \log p_i = 0$ and assume that $0 \log 0 = 0$.

We can interpret this equation as: *The uncertainty in a probability distribution is the average log-probability of an event* where an event could be weather, identity of a species of amphibian, or a particular amino acid residue.

Example: *Pseudomonas flourescens* bacteria have multiple phenotypes (Wrinkly Spreader (ws) vs. Smooth (sm)) with varying ecological success depending on whether they are in 2D or 3D environments (biofilm formation and all that; Spiers, 2005; 2007).

- If the phenotypic probabilities in a population are $p_{ws} = 0.75$ and $p_{sm} = 0.25$, respectively then

$$H(P) = -(p_{ws} \log p_{ws} + p_{sm} \log p_{sm}) \approx 0.56$$

- If a different population has $p_{ws} = 0.95$ and $p_{sm} = 0.05$, $H(P) \approx 0.2$. because there is less uncertainty about phenotypic expression; thus the entropy is less.

If we add a third phenotype (Big Wheel! (bw)) with probabilities $p_{ws} = 0.25$, $p_{sm} = 0.5$, and $p_{bw} = 0.25$, then entropy has increased to $H(P) \approx 1.04$.

These entropies themselves are not that useful to think about and are difficult to interpret individually. Instead, it makes more sense to compare them to other entropies.

Divergence

So now we can precisely say how hard it is to hit our target. Now we need to quantify how far our model is from said target using *Kulback-Leibler divergence.*

- *Divergence is the additional uncertainty introduced from using probabilities from one distribution to describe another distribution.*

Let's say that the true distribution of phenotypes is $p_{ws} = 0.3$ and $p_{bw} = 0.7$. But, we believe these events happen with probabilities Q: $q_{ws} = 0.25$ and $q_{bw} = 0.75$.

How much uncertainty have we introduced by using $Q = \{q_{ws}, q_{bw}\}$ to approximate $P = \{p_{ws}, p_{bw}\}$?

- If events occur according to $p's$ but are expected according to $q's$, the entropies are inflated depending on how different P and Q are. This inflation is called *cross entropy*: $H(P, Q) = -\sum_i p_i \log q_i$.
- Divergence is the *additional entropy* induced by using Q, so it's the difference between the actual entropy of events, $H(P)$, and the inflation of using Q to predict P, $H(P, Q)$:

$$\begin{aligned} D_{KL}(P, Q) &= H(P, Q) - H(P) \\ &= -\sum_i p_i \log q_i - (-\sum_i p_i \log p_i) \\ &= \sum_i p_i (\log p_i - \log q_i) \\ &= \sum_i p_i \log(\frac{p_i}{q_i}), \end{aligned}$$

- i.e., *the divergence is the average difference in log probability between the target (P) and model (Q).*

When $P = Q$, we know the actual probabilities of events and

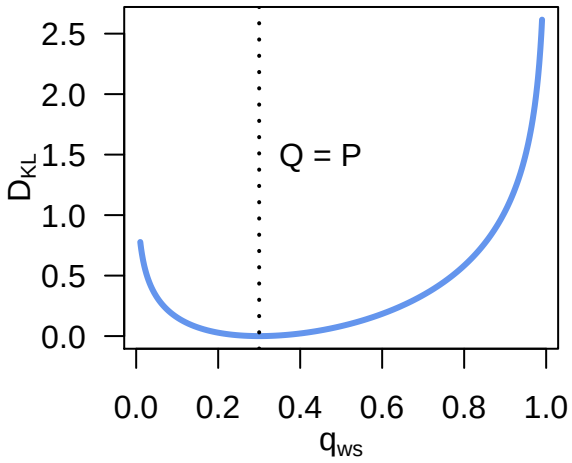
$$D_{KL}(P, Q) = D_{KL}(P, P) = \sum_i p_i (\log p_i - \log p_i) = 0$$

As Q grows more different than P, the divergence D_{KL} also grows. Back to our example where the true distribution is $P = \{0.3,0.7\}$:

- Lets say that our approximating distribution Q can range from $Q = \{0.01,0.99\}$ to $Q = \{0.99,0.01\}$. We can plot out the relationship between our first value ($q_w s$) and D_{KL}.

```
P <- c(0.3,0.7) # true target probs
ws <- seq(0.01,0.99, by=0.001) # probs for wrinkly spreader
Q <- cbind(ws, rev(ws)) # matrix of candidate values

KL <- function (P, Q) { # function to calculate divergence
  kl <- vector(length=length(P))
  for(i in 1:length(P)) {
    kl[i] <- P[i]*(log(P[i])-log(Q[i]))
  }
  return(sum(kl))
}

kl <- apply(Q, 1, KL, P=P) # calculate divergence for each row of Q
par(mar=c(3,3.2,0.1,0.5))
plot(Q[,1], kl, las=1, col="cornflowerblue", type="l", lwd=3)
mtext(text = expression(q[ws]), side=1, line=2)
mtext(text = expression(D[KL]), side=2, line=2.1)
```

```r
abline(v=0.3, lty=3, lwd=2) # where Q=P
text(0.45, 1.5, "Q = P")
```



Only when $Q = P = \{0.3, 0.7\}$ does $D_{KL} = 0$; everywhere else it grows. However, we can compare approximating functions to P.

- More accurate functions will shrink D_{KL}. THerefore we want the candidate closest to the target that minimizes the divergence.
- Predictive models quantify probabilities of events, so we can use diverence to compare model accuracy.

Deviance

So we know how to estimate the divergence from the true model P *IF* we know P before hand. But if we know the true model, there is no point to conducting statistical inference.

Luckily, we only need be concerned with comparing the divergences of different candidates, say Q and R.

- While we don't know P, we can estimate how far apart Q and R are.

Therefore we only need to know $E \log q_i$ and $E \log r_i$ for Q and R, respectively. If we assume the log probabilities provide an approximation of $E \log q_i$, we don't need P inside the expectation.

We can compare the average log-probability from each canddiate model to get estimates of the *relative* differences of each model from the target.

- The absolute magnitude of the values is not interpretable—neither $E \log q_i$ or $E \log r_i$ suggest a good or bad model by themselves.
- Only the difference $E \log q_i - E \log r_i$ informs us about the divergence of each model from the target P.

An approximation of the K-L divergence (and thus *relative* model fit) is the *deviance*:

$$D(Q) = -2 \sum_i \log q_i$$

where q_i is the likelihood of observation i.The -2 is for historical reasons because, in many cases, the distribution of two deviances has a χ^2-distribution.

NOTE that the deviance is not divided by the number of cases to make it an average. It is summed across cases. This means it will scale with sample size.

where q_i is the likelihood of observation i.The -2 is for historical reasons because, in many cases, the distribution of two deviances has a χ^2-distribution.

Because we are in Bayes land, there is uncertainty about parameters; thus there will also be uncertainty about the deviance of a model.

- For any specific parameter values, there is a defined deviance. But a posterior distribution of parameter values begets a posterior distribution of deviances.

Deviance to out-of-sample

Deviance is a nice way to measure the distance of our model from the target. But deviance has the same problem as R^2—it always improves as the model becomes more complex (this will be relaxed when we revisit multi-level modeling).

- Deviance, like R^2, is a measure of retrodictive accuracy, not predictive accuracy.
- What we want is the deviance on new data.

Enter in vs. out of sample: Imagine we have data and use that data to fit a statistical model.

- The data comprise the *training sample*. Parameters are estimated, and then we can imagine using those estimates to predict new outcomes from a new sample, called a *test sample*.

Here is an outline of a thought exercise to explore the distinction between deviance in and out of sample:

1. We have a training sample of size N.
2. Using the training sample, we fit a statistical model and compute the deviance—D_{train}.
3. Now imagine we have another sample of size N from the same population/biological process—the test sample.
4. We will now compute the deviance on the test sample (D_{test}) using our parameter estimates from step 2.

To visualize the results, we will conduct the thought experiment 500 times for five different regression models. The generating model is

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, 1) \\ \mu_i &= 0 + 0.15x_{1,i} - 0.4x_{2,i}. \end{aligned}$$

This is a Gaussian outocome y with an intercept $\alpha = 0$ and slopes for two predictors equaling $\beta_1 = 0.15$ and $\beta_2 = -0.4$, respectively.

- The first model will have one free parameter, a linear regression with an unknown mean and $\sigma = 1$.

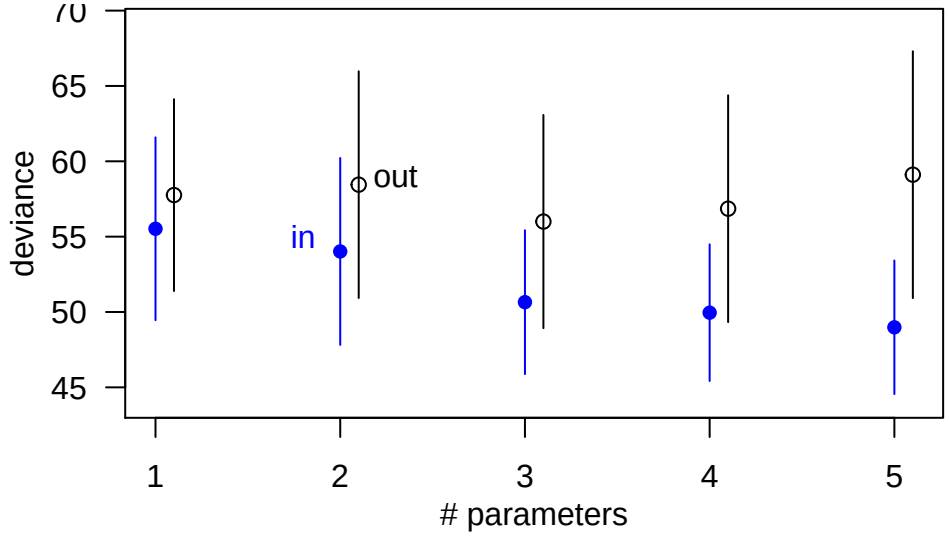
- Each parameter added adds a predictor variable and beta coefficient.
- The "true" model has non-zero coefficients for only the first two predictors, so the true model has 3 parameters total.

We will run these models using the `sim.train.test` function from the rethinking package. Note this takes a few minutes.

- I tried to streamline the code and add it to our `utility.functions.R` script but it took too long.

```
N <- 20
kseq <- 1:5
nSims <- 500

# If you have a mac set this
cores <- 4

dev <- sapply(kseq, function (k) {
#  print(k);
# If you have a mac use this
  r <- mcreplicate( nSims, sim.train.test(N=N, k=k), mc.cores=cores)
# If you don't have a mac use this
#  r <- replicate(nSims, sim.train.test( N=N, k=k ));
  c( mean(r[1,]), mean(r[2,]), sd(r[1,]), sd(r[2,]) )
} )
```

```
par(mar=c(3,3.2,0.1,0.5))
plot(1:5, dev[1,], ylim=c(min(dev[1:2,])-5, max(dev[1:2,]+10)),
  xlim=c(1,5.1), las=1, ann=FALSE, pch=16, col="blue")

mtext(text = "# parameters", side=1, line = 2, cex=1)
mtext(text = "deviance", side=2, line = 2.2, cex=1)

points((1:5)+0.1, dev[2,])
for (i in kseq) {
  IN <- dev[1,i] + c(-1,1)*dev[3,i]
  OUT <- dev[2,i] + c(-1,1)*dev[4,i]
  lines(c(i,i), IN, col="blue")
  lines(c(i,i)+0.1, OUT)
}

text(1.8, 55, "in", col="blue")
text(2.3, 59, "out")
```



In the plot, the blue points/lines show the mean $\pm 1SD$ of the deviance calculated on the training data. Increasing the number of parameters decreases the average deviance.

- Smaller deviances mean a better fit.

But look at the open points and black lines—the out-of-sample deviance.

- While the training deviance gets better with additional parameters, the average test deviance is smallest for three parameters, i.e., the data generating model.
  - as parameters are added, the deviance gets worse.

Note that there is no guarantee that the "true" model will have the smallest average out-of-sample deviance.

- This is illustrated by the 2 parameter model, which does worse on average than the one-parameter model. This is because there is uncertainty with only 20 samples.

Deviance is a measure of predictive accuracy, not truth. The true model is not guaranteed to produce the best predictions, and a false model is not guaranteed to produce bad predictions.