

Topics in Multivariate Analysis

APSTA GE-2004

Lecture 5 - Information Criteria, CV, and Model Evaluation

2/22/2022

Outline

Welcome! Today, we'll cover the following:

- Information criteria
- Cross validation
- Model evaluation and comparison

Reading:

RAOS Ch 7.2; Ch 11.6-11.8

Information Criteria

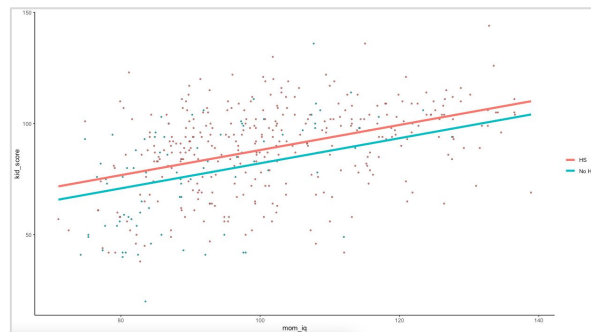
Information criteria

At this point, we've learned how to generate a variety of linear (additive) models for a dataset

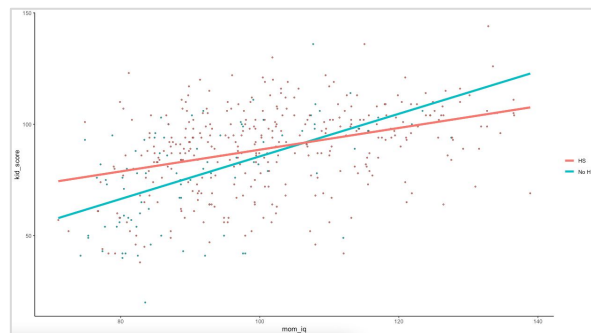
For example, we have a varying-intercepts model (top) and a varying-intercepts, varying-slopes model (bottom) for the kid score dataset

How can we evaluate and compare these models in a principled way? Why might we favor one of these models over the other?

Model with varying intercepts



Model with varying intercepts & varying slopes (interaction)



Information criteria

Here are the steps we will use to be able to evaluate and compare these models in a principled way:

1. Establish a measurement scale for distance from perfect accuracy
2. Establish *deviance* as an approximation of relative distance from perfect accuracy
3. Establish that it is only deviance *out-of-sample* that is of interest

Measuring accuracy: Joint probability

Accuracy depends on the definition of the target. In defining a target, there are two major dimensions:

- *Cost-benefit analysis*: How much does it cost when we're wrong? How much do we win when we're right?
- *Accuracy in context*: Some prediction tasks are easier than others











Average vs Joint

Which definition of "accuracy" is maximized by knowing the true model generating the data?

Joint probability

Joint probability is the definition of accuracy that we want because it is maximized by the correct model. It's the unique measure that correctly counts up the relative number of ways each event could happen











Current weather person

Day	1	2	3	4	5	6	7	8	9	10
Prediction	1	1	1	0.6	0.6	0.6	0.6	0.6	0.6	0.6
Observed										

Average: $3 * 1 + 7 * 0.4 = 5.8$ hits in 10 days, or "hit rate" of $5.8/10 = 0.58$ correct predictions per day

Joint probability of being correct: $1^3 * 0.4^7 = 0.005$

Newcomer

Day	1	2	3	4	5	6	7	8	9	10
Prediction	0	0	0	0	0	0	0	0	0	0
Observed										

Average: $3 * 0 + 7 * 1 = 7$ hits in 10 days, or "hit rate" of $7/10 = 0.7$ correct predictions per day

Joint probability of being correct: $0^3 * 1^7 = 0$ (newcomer has zero probability of getting the sequence correct)

Information entropy

Information: the reduction in uncertainty when we learn an outcome

Properties a measure of uncertainty should possess:

1. The measure of uncertainty should be continuous
2. The measure of uncertainty should increase as the number of possible events increases
3. The measure of uncertainty should be additive

There is only one function that satisfies these properties:

$$H(p) = -E \log(p_i) = -\sum p_i \log(p_i)$$

Information entropy: the uncertainty contained in a probability distribution is the average log-probability of an event

A weather example:

Suppose in one location the true probabilities of rain and shine are $p_1 = 0.3$ and $p_2 = 0.7$, respectively, then the information entropy for the weather is:

```
> p <- c( 0.3 , 0.7 )  
> round( -sum( p*log(p) ) , 3 )  
[1] 0.611
```

In another location the true probabilities of rain and shine are $p_1 = 0.01$ and $p_2 = 0.99$, respectively, then the information entropy for the weather is:

```
> p <- c( 0.01 , 0.99 )  
> round( -sum( p*log(p) ) , 3 )  
[1] 0.056
```

Divergence

Information entropy gives us a way to quantify uncertainty (to say, in a precise way, how hard it is to hit the target)

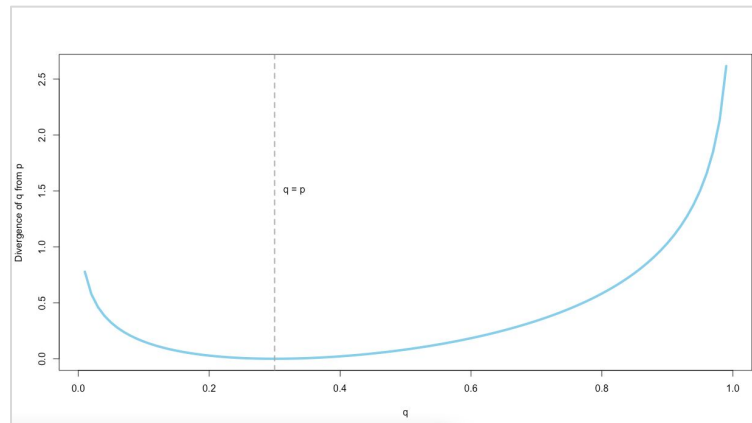
But how can we use it to say how far a model is from the target? The key lies in *divergence*:

Divergence: the additional uncertainty induced by using probabilities from one distribution to describe another distribution

$$D_{KL}(p, q) = \sum p_i (\log(p_i) - \log(q_i)) = \sum p_i \log(p_i / q_i)$$

That is, the divergence is the average difference in log probability between the target (p) and model (q)

This is also known as *Kullback-Leibler divergence* or simply *KL divergence*



As q grows more different from p , the divergence D_{KL} also grows:

Suppose the true target distribution is $p = \{0.3, 0.7\}$ and the approximating distribution q can be anything from $q = \{0.01, 0.99\}$ to $q = \{0.99, 0.01\}$.

As shown above, only exactly where $q = p$, at $q_1 = 0.3$, does the divergence achieve a value of zero. Everywhere else, it grows

From Divergence to Deviance

The preceding material establishes how to measure the distance of a model from our target – the distance measure we need is *KL divergence*

Having identified the right measure of distance, now we need a way to estimate it – divergence leads us to using a measure of model fit known as *deviance*

To use D_{KL} to compare models, it seems like we would have to know p , the target probability distribution, but we don't need to because we are only interested in comparing the divergences of different candidates, say q and r

In this case, most of p subtracts out because there's a $E \log(p_i)$ term in the divergence of both q and r . So while we don't know where p is, we can estimate how far apart q and r are, and which is closer to the target

All we need to know is a model's average log-probability:

$$E \log(q_i) \text{ for } q \quad \text{and} \quad E \log(r_i) \text{ for } r$$

Log Pointwise Predictive Density and Deviance

Log-Probability Score

To put this into practice, its conventional to sum over all the observations i , yielding a total score for a model q :

$$S(q) = \sum \log(q_i)$$

We can compare the average log-probability from each model to get an estimate of the relative distance of each model from the target. The absolute magnitude of these values will not be interpretable. Only the difference $E \log(q_i) - E \log(r_i)$ informs us about the divergence of each model from the target p

Log-Pointwise Predictive Density

For a Bayesian model, we have to use the entire posterior distribution. Doing so, we find the ***lppd***, the log of the average probability for each observation i , where the average is taken over the posterior distribution. Larger values are better, because that indicates larger average accuracy

Deviance

Like a lppd score, but multiplied by -2 so smaller values are better. The 2 is there for historical reasons

Log Pointwise Predictive Density (lppd)

The Bayesian version of the log-probability score is called the log-pointwise predictive density (*lppd*)

	species	brain mass	mass_std	brain_std
1	afarensis	438 37.0	-0.7794667	0.3244444
2	africanus	452 35.5	-0.9170196	0.3348148
3	habilis	612 34.5	-1.0087216	0.4533333
4	boisei	521 41.5	-0.3668079	0.3859259
5	rudolfensis	752 55.5	0.9170196	0.5570370
6	ergaster	871 61.0	1.4213804	0.6451852
7	sapiens	1350 53.5	0.7336157	1.0000000

```
m7.1 <- quap(
  alist(
    brain_std ~ dnorm( mu , exp(log_sigma) ),
    mu <- a + b*mass_std,
    a ~ dnorm( 0.5 , 1 ),
    b ~ dnorm( 0 , 10 ),
    log_sigma ~ dnorm( 0 , 1 )
  ), data=d )
precis( m7.1 )
```

```
> lppd( m7.1 , n=1e4 )
[1] 0.6098669 0.6483439 0.5496093 0.6234935 0.4648144 0.4347605 -0.8444634
>
> sum( lppd( m7.1 , n=1e4 ) )
[1] 2.471056
```

```
# extract posterior draws of parameters
# and set up constant variables
posterior <- extract.samples( m7.1 )
y <- d$brain_std
N <- length(y)
S <- nrow(posterior)
loglik <- yloo <- sdloo <- matrix(nrow = S, ncol = N)

# calculate the log-probability of each observation
# return a matrix with a row for each sample
# and a column for each observation
set.seed(1)
for (s in 1:S) {
  p <- posterior[s, ]
  yloo[s, ] <- p$a + p$b * d$mass_std
  sdloo[s, ] <- exp(p$log_sigma)
  loglik[s, ] <- dnorm(y, yloo[s, ], sdloo[s, ], log = TRUE)
}

# calculate the total log-probability score for the model and data
# larger values are better, because that indicates larger average accuracy
log_mean_exp <- function(x) {
  # more stable than log(mean(exp(x)))
  max_x <- max(x)
  max_x + log(sum(exp(x - max_x))) - log(length(x))
}

( lppds <- apply(loglik , 2 , log_mean_exp) )

sum( lppds )
```

```
> ( lppds <- apply(loglik , 2 , log_mean_exp) )
[1] 0.6098669 0.6483438 0.5496093 0.6234934 0.4648144 0.4347605 -0.8444633
>
> sum( lppds )
[1] 2.486425
```

Underfitting and overfitting

Under-sensitivity and over-sensitivity to same

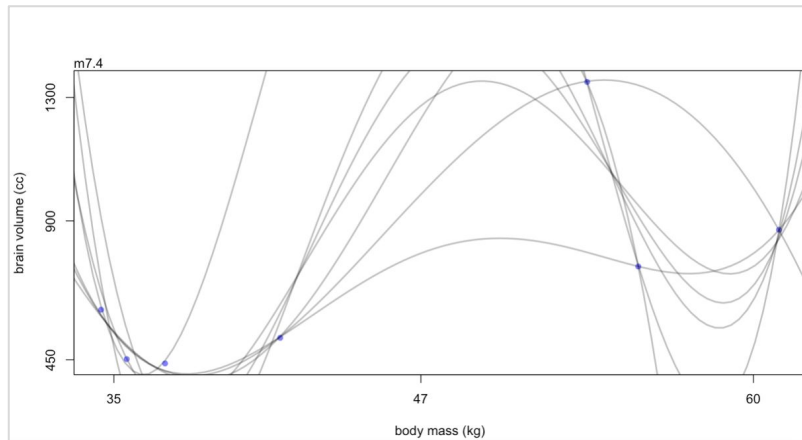
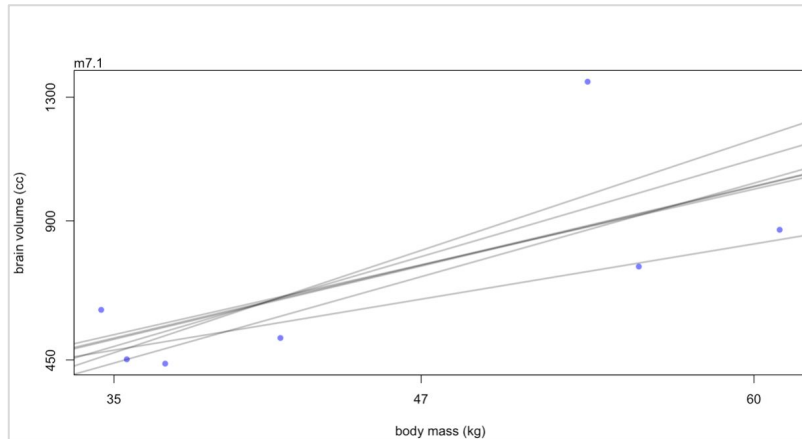
In both plots, a regression is fit to the seven sets of data made by dropping one row from the original data

Top: An underfit model is insensitive to the sample, changing little as individual points are dropped

Bottom: An overfit model is sensitive to the sample, changing dramatically as points are dropped

So how do we navigate between underfitting and overfitting?

Whether using regularization or information criteria or both, we need to focus on ***deviance out-of-sample***



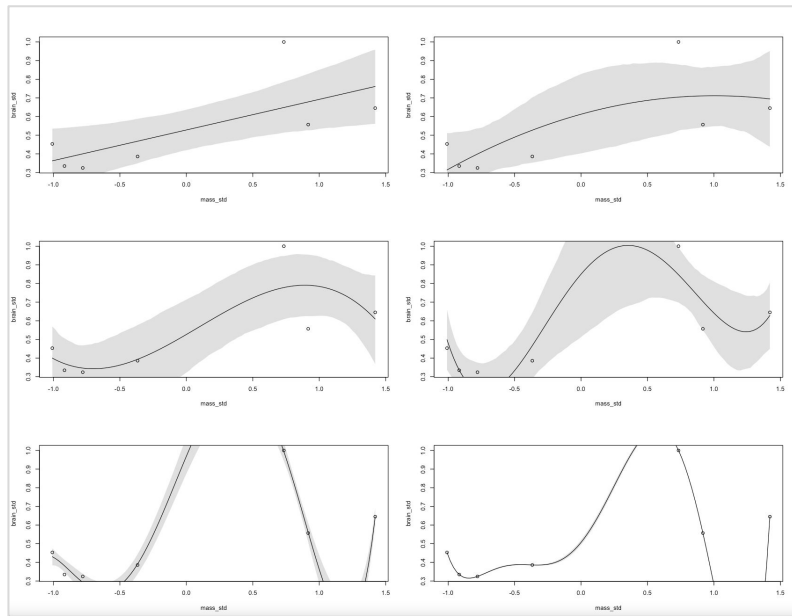
Score the right data (Out-of-sample)

The *log-probability* score (and *deviance*) is a principled way to measure distance from the target. But the score as computed has the same flaw as R^2 : *It always improves as the model gets more complex*

As the degree of the polynomial defining the mean increases, the R^2 always improves, indicating better retrodiction of the data (in fact, the sixth-degree polynomial passes through every point and has no residual variance...it's a perfect fit, $R^2 = 1$)

However, you can see from the paths of the predicted means that the higher-degree polynomials are increasingly absurd

We can not score models by their performance on training data. It is really the score on *new* data that interests us.



```
> sapply( list(m7.1,m7.2,m7.3,m7.4,m7.5,m7.6) , function(m) sum(lppd(m)) )  
[1] 2.490381 2.565984 3.707338 5.333662 14.094142 39.445390
```

Score the right data (Out-of-sample)

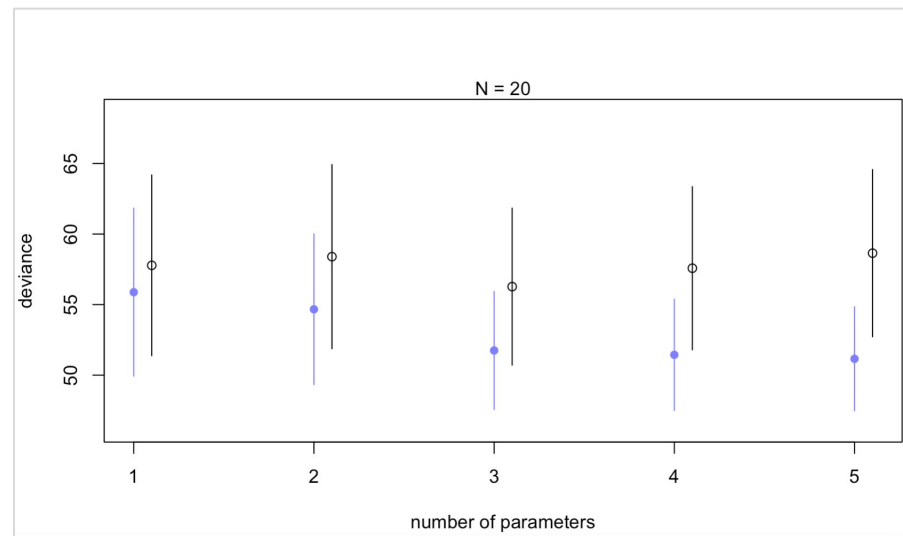
In the plot, models with different numbers of predictors are shown in the horizontal axis. Deviance across 10,000 simulations is shown on the vertical

Blue shows deviance *in-sample*, the *training* data. Black shows deviance *out-of-sample*, the *test* data. Points show means, and the line segments show ± 1 standard deviation

Since the “true” model has non-zero coefficients for only the first two predictors, we can say the true model has 3 parameters

Moving left to right, the blue points show average deviance declines (smaller deviance means better fit) in-sample

Conversely, the test deviance is smallest on average for 3 parameters, which is the data-generating model in this case



Regularizing priors

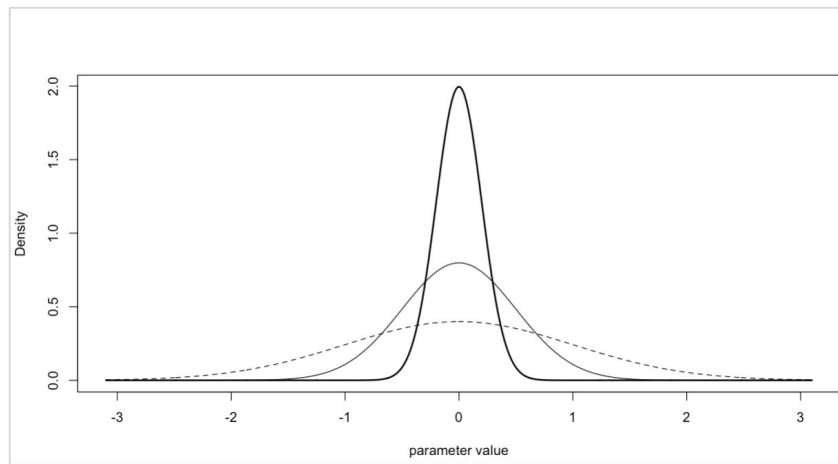
One way to produce better predictions is to make the model worse at fitting the sample

The root of overfitting is a model's tendency to get over-excited by the training sample

One way to prevent a model from getting over-excited by the training sample is to use a skeptical prior, a prior that slows the rate of learning from the sample

The figure shows three Gaussian priors of varying standard deviation: Dashed: $N(0, 1)$ Thin solid: $N(0, 0.5)$ Thick solid: $N(0, 0.2)$

When more probability is massed up around zero, estimates are shrunk towards zero – they are conservative



Regularizing priors

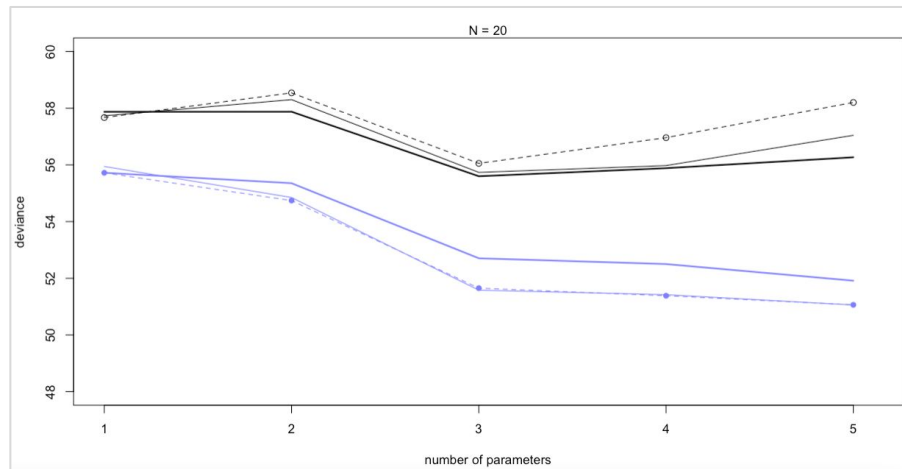
Regularizing priors and out-of-sample deviance

The points are the same as the previous out-of-sample plot. The lines show training (blue) and testing (black) deviance for the three regularizing priors

The **training** deviance always increases – gets worse – with tighter priors. The thick blue trend is substantially larger than the others because the skeptical prior prevents the model from adapting completely to the sample

But the **test** deviances, out-of-sample, improve – get smaller – with the tighter priors. The model with three parameters is still the best model out-of-sample

In addition, as the prior gets more skeptical, the harm done by an overly complex model is greatly reduced. If you can tune the regularizing prior right, then overfitting can be greatly reduced



Predicting predictive accuracy

The preceding suggest one way to navigate overfitting and underfitting is to evaluate our models out-of-sample, but we do not have the out-of-sample, by definition, so how can we evaluate our models on it?

There are two families of strategies:

- Cross-validation
- Information criteria

These strategies try to guess how well models will perform, on average, in predicting new data

Cross Validation

Cross-validation

A popular strategy for estimating predictive accuracy is to leave out a small chunk of observations from our sample and evaluate the model on the observations that were left out

To do so, divide the sample into a number of chunks, called “folds”, predict each fold after training on the other folds, and average over all the scores

The minimum number of folds is 2. At the other extreme, you could make each observation a fold and fit as many models as you have individual observations

Leave-one-out cross-validation (LOOCV): it is common to use the maximum number of folds, resulting in leaving out one unique observation in each fold

Pareto-smoothed importance sampling cross-validation (PSIS)

One approach to approximate leave-one-out cross-validation is to use the “importance” of each observation to the posterior distribution

PSIS uses these “importance” weights, along with Pareto-smoothing to make the importance weights more reliable, to derive a more reliable cross-validation score without actually doing any cross-validation

The best feature of PSIS is that it provides feedback about its own reliability by noting particular observations with very high weights that could make the PSIS score inaccurate

Another nice feature is that it is computed point by point, which provides an approximate estimate of the standard error of the estimate of out-of-sample deviance

PSIS-LOO cross validation

Use simulated data and choose a point with a large residual to see where cross validation can make a difference

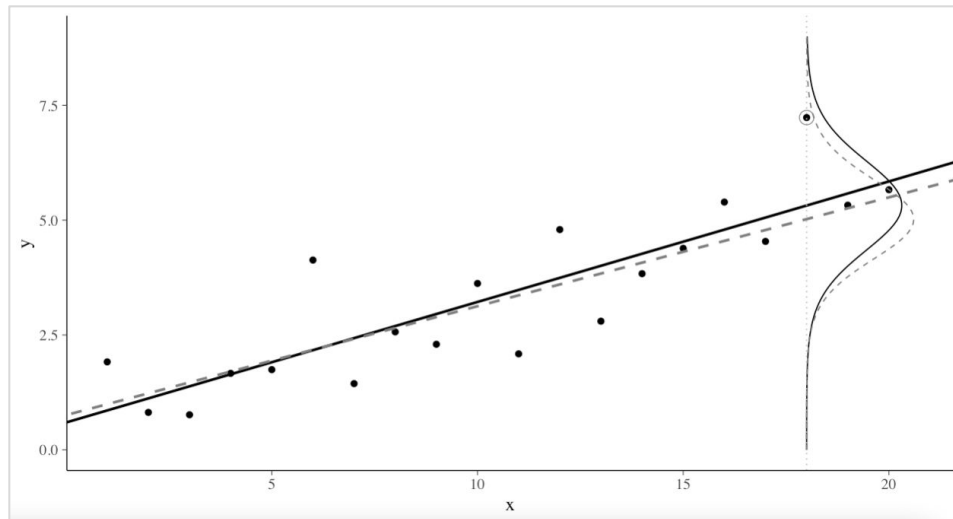
Fit two models:

1. Model fit to all the data
2. Model fit to the data excluding the one with a large residual

Evaluate the prediction for the data point with a large residual

The fitted regression lines reveal that the fit when excluding the data point is different from the fit from all the data

When a data point is held out, the posterior mean and predictive distribution move away from that observation



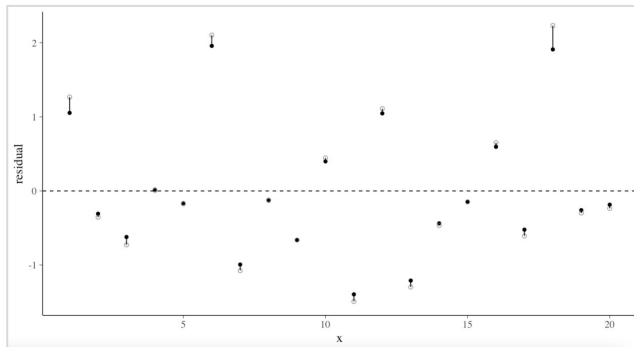
PSIS-LOO cross validation

Use simulated data and choose a point with a large residual to see where cross validation can make a difference

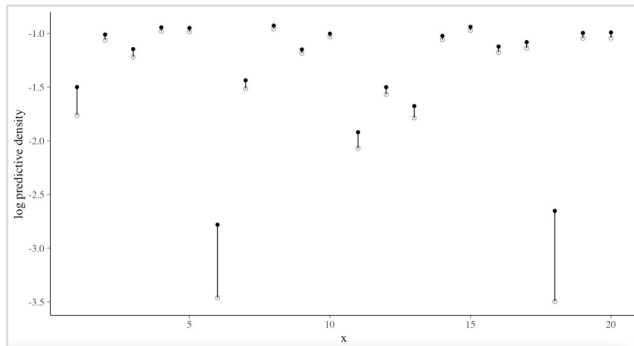
Fit two models:

1. Model fit to all the data
2. Model fit to the data excluding the one with a large residual

Top: Residuals from the fitted model. Solid circles show residuals from the model fit to all the data; open circles show leave-one-out residuals, where each observation is compared to its expected value from the model holding that data point out of the fit



Bottom: Log posterior predictive densities and log LOO predictive from the models. Solid circles represent log predictive densities relative to the model fit to all the data; open circles correspond to leave-one-out fits. The open circles are always lower than the solid circles, because removing an observation from the likelihood causes the model to fit less well to that data point



PSIS-LOO cross validation

Interpreting the **loo** output:

- The “log-likelihood matrix” is $\log p(y_i | x_i, \beta, \sigma)$ at $n = 434$ data points (x_i, y_i) using 4000 draws of β and σ from the posterior distribution
- **elpd_loo** is the estimated log score along with a standard error representing uncertainty due to using only 434 data points
- **p_loo** is the estimated “effective number of parameters” in the model. The model has 4 parameters, so it makes sense that **p_loo** is close to 4 here, but in models with stronger priors, weaker data, or model misspecification, this direct identification will not always work
- **looic** is the LOO information criterion, -2elpd_loo , for comparability to deviance
- The **Monte Carlo SE of elpd_loo** gives the uncertainty of the estimate based on the posterior simulations. If the number of simulations is large enough, this Monte Carlo SE will approach zero, in contrast to the standard errors of **elpd_loo**, **p_loo**, and **looic**, whose uncertainties derive from the finite number of data points in the regression
- The two lines at the end of the display give a warning if the Pareto k estimates are unstable, in which case an alternative computation is recommended

```
fit_3 <- stan_glm(kid_score ~ mom_hs + mom_iq, data = kidiq)
loo_3 <- loo::loo(fit_3)
print(loo_3)
```

```
> loo_3 <- loo::loo(fit_3)
> print(loo_3)
```

Computed from 4000 by 434 log-likelihood matrix

	Estimate	SE
elpd_loo	-1876.2	14.2
p_loo	4.1	0.4
looic	3752.4	28.4

Monte Carlo SE of elpd_loo is 0.0.

All Pareto k estimates are good ($k < 0.5$).
See `help('pareto-k-diagnostic')` for details.

K-fold cross validation

The **loo** function has a computationally-efficient approximation to LOOCV that requires the model to be fit only once, but in some cases it can be unstable

Difficulties arise when the probability densities $p(y_i|\theta)$ take on very low values – if this density is near zero, then the resulting weight, $1/p(y_i|\theta)$, becomes very high

When *loo* cross validation is unstable – which is indicated by a warning message – you can switch to *K-fold cross validation* in which the data are randomly partitioned into K subsets, and the fit of each subset is evaluated based on a model fit to the rest of the data. This requires K new model fits

It is conventional to use $K = 10$, which represents tradeoff between K being *too low* (in which case the estimate has larger bias) or *too high* (then the procedure becomes too computationally expensive)

```
> fit_1 <- stan_glm(y ~ ., prior=normal(0, 10, autoscale=FALSE),  
+ data=fake, seed=SEED, refresh=0)  
> ( loo_1 <- loo(fit_1) )
```

Computed from 4000 by 60 log-likelihood matrix

	Estimate	SE
elpd_loo	-153.0	5.1
p_loo	27.9	3.3
looc	306.1	10.2

Monte Carlo SE of elpd_loo is NA.

Pareto k diagnostic values:

	Count	Pct.	Min.	n_eff
(-Inf, 0.5]	(good)	15	25.0%	919
(0.5, 0.7]	(ok)	32	53.3%	237
(0.7, 1]	(bad)	13	21.7%	48
(1, Inf)	(very bad)	0	0.0%	<NA>

See `help('pareto-k-diagnostic')` for details.

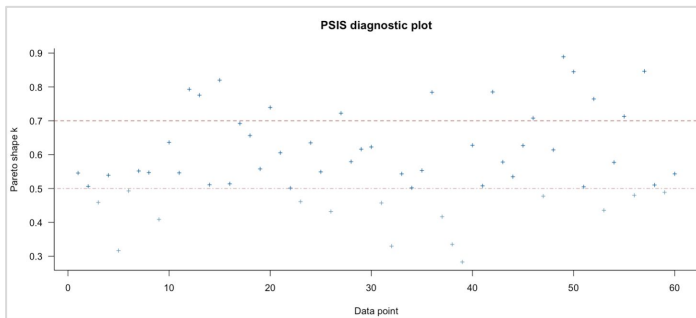
Warning message:

Found 13 observations with a `pareto_k` > 0.7. With this many problematic observations we recommend calling 'kfold' with argument 'K=10' to perform 10-fold cross-validation rather than LOO.

```
> # In this case, Pareto smoothed importance sampling LOO fails,  
> # but the diagnostic recognizes this with many high Pareto k values.  
> # We can run slower, but more robust K-fold-CV  
> ( kfold_1 <- kfold(fit_1, K=10) )  
Fitting K = 10 models distributed over 16 cores
```

Based on 10-fold cross-validation

	Estimate	SE
elpd_kfold	-158.6	5.8
p_kfold	NA	NA
kfoldic	317.3	11.5



Information Criteria

Information criteria

Another strategy for estimating predictive accuracy is to use *information criteria* to compute an expected score out-of-sample

Information criteria construct a theoretical estimate of the relative out-of-sample KL divergence

If you look back at the in-sample, out-of-sample plots, you'll notice a curious pattern – *the difference between the train-test points is approximately twice the number of parameters in each model*

This is not a coincidence but rather a cool result in machine learning – *for ordinary linear regressions with flat priors, the expected overfitting penalty is about twice the number of parameters*

Akaike Information Criterion (AIC)

AIC provides a simple estimate of the average out-of-sample deviance:

$$\text{AIC} = D_{\text{train}} + 2p = -2 \cdot \text{lppd} + 2p$$

where p is the number of parameters in the posterior distribution

As the 2 is just there for scaling, AIC tells us the dimensionality of the posterior distribution is a natural measure of the model's overfitting tendency. *More complex models tend to over fit more, directly in proportion to the number of parameters*

AIC is an approximation that is reliable only when:

- The priors are flat or overwhelmed by the likelihood
- The posterior distribution is approximately multivariate Gaussian
- The sample size N is much greater than the number of parameters k

Since flat priors are hardly ever the best priors, we'll want something more general

There is a more general criterion, the *Deviance Information Criterion (DIC)*, which is okay with informative priors, but still assumes the posterior is multivariate Gaussian and that $N \gg k$

Widely Applicable Information Criterion (WAIC)

WAIC is more general than both AIC and DIC

It makes no assumption about the shape of the posterior

It provides an approximation of the out-of-sample deviance that converges to the cross-validation approximation *in a large sample*

However, *in a finite sample*, it can disagree because it has a different target – it isn't trying to approximate the cross-validation score, but rather guess the *out-of-sample KL divergence*

WAIC is the log-posterior-predictive-density (lppd), plus a penalty proportional to the variance in the posterior predictions:

$$\text{WAIC} = -2 * (\text{lppd} - \sum \text{var}_{\theta} \log p(y_i | \theta))$$

where y is the observations and θ is the posterior distribution

The penalty term (*everything after the minus sign*) mean, “compute the variance in log-probabilities for each observation i , and then sum up these variances to get the total penalty.” So you can think of each observation as having its own personal penalty score

Since these scores measure overfitting risk, you can also assess overfitting at the level of each observation

Widely Applicable Information Criterion (WAIC)

Like PSIS, WAIC is *pointwise*. Prediction is considered case-by-case, or point-by-point, in the data

Several things arise from this:

- First, WAIC also has an approximate standard error
- Second, since some observations have stronger influence on the posterior distribution, WAIC notes this in its pointwise penalty terms
- Third, like cross-validation and PSIS, because WAIC allows splitting up the data into independent observations, it is sometimes hard to define (e.g. in a time series)

```
# To see how the WAIC calculations work, consider the following regression:
data(cars)
m <- quap(
  alist(
    dist ~ dnorm(mu,sigma),
    mu <- a + b*speed,
    a ~ dnorm(0,100),
    b ~ dnorm(0,10),
    sigma ~ dexp(1)
  ), data=cars )
set.seed(94)
post <- extract.samples(m, n=1000)

# We'll need the log-likelihood of each observation i
# at each sample s from the posterior:
n_samples <- 1000
logprob <- sapply( 1:n_samples ,
  function(s) {
    mu <- post$a[s] + post$b[s]*cars$speed
    dnorm( cars$dist, mu, post$sigma[s] , log=TRUE )
  } )

# You end up with a 50-by-1000 matrix of log-likelihoods, with observations
# in rows and samples in columns

# Now to compute lppd, the Bayesian deviance, we average the samples
# in each row, take the log, and add all of the logs together
# To do this with precision, you need to do the averaging on the log scale
n_cases <- nrow(cars)
lppd <- sapply( 1:n_cases , function(i) log_sum_exp(logprob[i,]) - log(n_samples) )
# Typing 'sum(lppd)' will give you lppd.

# Now for the penalty term, pWAIC, we compute the variance across samples
# for each observation, then add these together:
pWAIC <- sapply( 1:n_cases , function(i) var(logprob[i,]) )
# Typing 'sum(pWAIC)' will give you the penalty term

# Finally, the following computes WAIC:
-2*( sum(lppd) - sum(pWAIC) )

# Compute the standard error of WAIC by computing the square root
# of the number of cases multiplied by the variance over the
# individual observation terms in WAIC:
waic_vec <- -2*( lppd - pWAIC )
sqrt( n_cases*var(waic_vec) )
```

WAIC metaphors

Think of models as race horses:

- In any particular race, the best horse may not win. But it's more likely to win than is the worst horse
- And when the winning horse finishes in half the time of the second-place horse, you can be pretty sure the winning horse is also the best. But if instead it's a photo-finish, with a near tie between first and second place, then it is much harder to be confident about which is the best horse

WAIC values are analogous to these race times – smaller values are better, and the distances between the horses/models are informative

Think of models as stones thrown to skip on a pond:

- No stone will ever reach the other side (*perfect prediction*), but some sorts of stones make it farther than others, on average (*make better test predictions*)
- But on any individual throw, lots of unique conditions avail – the wind might pick up or change direction, a duck could surface to intercept the stone, or the thrower's grip might slip. So which stone will go farthest is not certain

Still, the relative distances reached by each stone provide information about which stone will do best on average

But we can't be too confident about any individual stone, unless the distances between stones is very large

Model Comparison

Model comparison

When there are several plausible (and hopefully un-confounded) models for the same set of observations, *how should we compare the accuracy of these models?*

Following the fit to the sample is no good, because fit will always favor more complex models

Information divergence is the right measure of model accuracy, but even it will just lead us to choose more and more complex and wrong models

We need to somehow evaluate models out-of-sample

Regularizing priors and CV / PSIS / WAIC are complementary. Regularization reduces overfitting, and predictive criteria measure it

Model comparison: Demonstration of adding pure noise predictors to a model

When predictors are added to a model – even if the predictors are pure noise – *the fit to data and **within-sample** predictive measures will generally improve:*

- R^2 *increases* from 0.21 to 0.23 (increase of 0.02)
- The posterior predictive log score *increases* from -1872 to -1869 (increase of 3)

This pattern, that fit to data improves even with noise predictors, is an example of overfitting, and it is one reason we have to be careful when throwing predictors into a regression

Using **LOO CV** we see that *the predictive performance for **new data** or generalization capability has not improved:*

- LOO R^2 *decreases* from 0.2 to 0.19 (decrease of 0.01)
- The LOO posterior predictive log score *decreases* from -1876 to -1878 (decrease of 2)

```
> fit_3 <- stan_glm(kid_score ~ mom_hs + mom_iq, data=kidiq,
+                 seed=SEED, refresh = 0)
> fit_3n <- stan_glm(kid_score ~ mom_hs + mom_iq + noise, data=kidiq,
+                 seed=SEED, refresh = 0)
>
> round( R2 <- 1 - var(respost)/var(kidiq$kid_score) , 2 )
[1] 0.21
> round( R2n <- 1 - var(respostn)/var(kidiq$kid_score) , 2 )
[1] 0.23
>
> round( sum( matrixStats::colLogSumExps(ll_3) - log(nrow(ll_3)) ) , 1 )
[1] -1872
> round( sum( matrixStats::colLogSumExps(ll_3n) - log(nrow(ll_3n)) ) , 1 )
[1] -1869.1
>
>
> round( R2loo <- 1 - var(resloo)/var(kidiq$kid_score) , 2 )
[1] 0.2
> round( R2loon <- 1 - var(resloon)/var(kidiq$kid_score) , 2 )
[1] 0.19
>
> round( loo_3$estimate["elpd_loo",1] , 1 )
[1] -1876.1
> round( loo_3n$estimate["elpd_loo",1] , 1 )
[1] -1878
```

Model comparison: Comparing two kidiq models

The simpler model performs worse in cross validation: elpd_{loo} has decreased from -1876.1 to -1914.9 , a decline of 38.8

To get a sense of the uncertainty in this improvement for future data, we need to compare the two models directly for each data point, which can be done using the ***loo_compare*** function:

The difference of 38.3 has a standard error of only 8.3

When the elpd_diff is larger than 4 , the number of observations is larger than 100 , and the model is not badly misspecified then the standard error (se_diff) is a reliable measure of the uncertainty in the difference. Differences smaller than 4 are hard to distinguish from noise

```
> # Model using only the maternal high school indicator
> fit_1 <- stan_glm(kid_score ~ mom_hs, data=kidiq, refresh = 0)
> ( loo_1 <- loo(fit_1) )
```

Computed from 4000 by 434 log-likelihood matrix

	Estimate	SE
elpd_loo	-1914.9	13.8
p_loo	3.2	0.3
looic	3829.8	27.7

Monte Carlo SE of elpd_loo is 0.0.

All Pareto k estimates are good ($k < 0.5$).
See help('pareto-k-diagnostic') for details.

```
>
> fit_3 <- stan_glm(kid_score ~ mom_hs + mom_iq, data=kidiq,
+                   seed=SEED, refresh = 0)
> ( loo_3 <- loo(fit_3) )
```

Computed from 4000 by 434 log-likelihood matrix

	Estimate	SE
elpd_loo	-1876.1	14.2
p_loo	4.0	0.4
looic	3752.2	28.4

Monte Carlo SE of elpd_loo is 0.0.

All Pareto k estimates are good ($k < 0.5$).
See help('pareto-k-diagnostic') for details.

```
>
> # Compare models using LOO log score (elpd)
> loo_compare(loo_3, loo_1)
      elpd_diff se_diff
fit_3    0.0      0.0
fit_1 -38.8     8.3
```

Model comparison: Comparing poisson vs negative binomial models for the roaches data

This [example](#) comes from Chapter 8.3 of Gelman and Hill (2007). We want to make inferences about the efficacy of a certain pest management system at reducing the number of roaches in urban apartments

Model 1: Poisson regression (used for count data)

The $elpd_{loo}$ estimate should not be considered reliable (If we had a well-specified model we would expect the estimated effective number of parameters (p_{loo}) to be smaller than or similar to the total number of parameters in the model. Here p_{loo} is almost 300, which is about 70 times the total number of parameters in the model, indicating severe model misspecification)

Model 2: Negative binomial regression (used for overdispersed count data)

The *Monte Carlo SE* is small compared to the other uncertainties. On the other hand, p_{loo} is about 7 and still a bit higher than the total number of parameters in the model. This indicates that there is almost certainly still some degree of model misspecification, but this is much better than the p_{loo} estimate for the Poisson model

The difference in ELPD is much larger than several times the estimated standard error of the difference, indicating that the negative-binomial model is expected to have better predictive performance than the Poisson model

```
> loo1 <- loo(fit1, save_psis = TRUE)
Warning message:
Found 17 observations with a pareto_k > 0.7. With this many problematic observations we recommend calling 'kfold' with argument 'K=10'
to perform 10-fold cross-validation rather than LOO.

> print(loo1)

Computed from 4000 by 262 log-likelihood matrix

      Estimate      SE
elpd_loo -6247.8  728.0
p_loo     292.4   73.3
looic     12495.5 1455.9
-----
Monte Carlo SE of elpd_loo is NA.

Pareto k diagnostic values:
      Count Pct. Min. n_eff
(-Inf, 0.5] (good)  239  91.2%  200
(0.5, 0.7] (ok)     6   2.3%   56
(0.7, 1] (bad)      8   3.1%   25
(1, Inf] (very bad) 9   3.4%    1
See help("pareto-k-diagnostic") for details.
>
> loo2 <- loo(fit2, save_psis = TRUE)
Warning message:
Found 1 observation(s) with a pareto_k > 0.7. We recommend calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
the ELPD without the assumption that these observations are negligible. This will refit the model 1 times to compute the ELPDs for the
problematic observations directly.

> print(loo2)

Computed from 4000 by 262 log-likelihood matrix

      Estimate      SE
elpd_loo -895.6  37.8
p_loo      6.7   2.7
looic     1791.3  75.5
-----
Monte Carlo SE of elpd_loo is NA.

Pareto k diagnostic values:
      Count Pct. Min. n_eff
(-Inf, 0.5] (good)  260  99.2% 2615
(0.5, 0.7] (ok)     1   0.4%   377
(0.7, 1] (bad)      1   0.4%    28
(1, Inf] (very bad) 0   0.0%   <NA>
See help("pareto-k-diagnostic") for details.
>
> loo_compare(loo1, loo2)
      elpd_diff se_diff
fit2      0.0      0.0
fit1 -5352.1  709.2
```

Model comparison: High pareto k values? ...maybe switch to robust regression

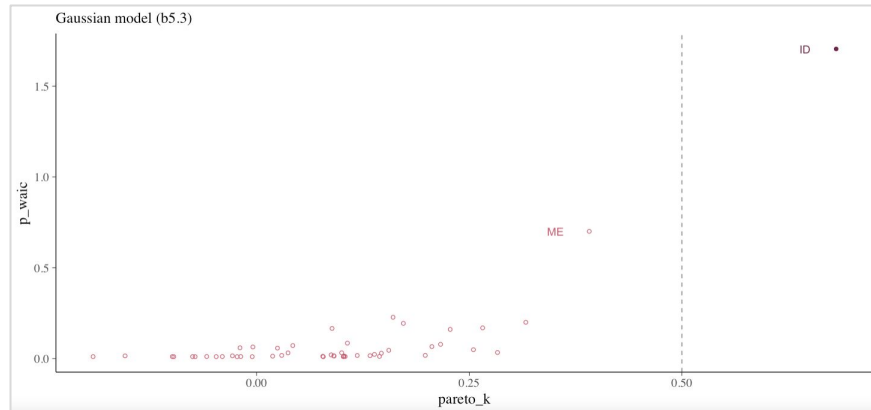
Comparing models predicting divorce rates, we receive a warning message that **Some Pareto k values are very high**, which means the smoothing approximation that PSIS-LOO uses is unreliable for some points

Looking at individual states, we can see the State of Idaho (ID) has an extremely unlikely value, according to the model. As a result, it is highly influential – it exerts more influence on the posterior distribution than other states

What can be done about this?

One way to both use these extreme observations and reduce their influence is to employ some kind of **robust regression** – *which usually means replacing the Gaussian model with a thicker-tailed distribution like Student's T (or “Student- t ”) distribution*

```
> rethinking::compare( m5.1 , m5.2 , m5.3 , func = PSIS )
Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.
Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.
Some Pareto k values are very high (>1). Set pointwise=TRUE to inspect individual points.
      PSIS   SE dPSIS  dSE pPSIS weight
m5.1 127.6 14.69   0.0  NA   4.7  0.83
m5.3 130.8 16.16   3.2  1.81  6.6  0.17
m5.2 141.2 11.57  13.7 10.92  4.1  0.00
```



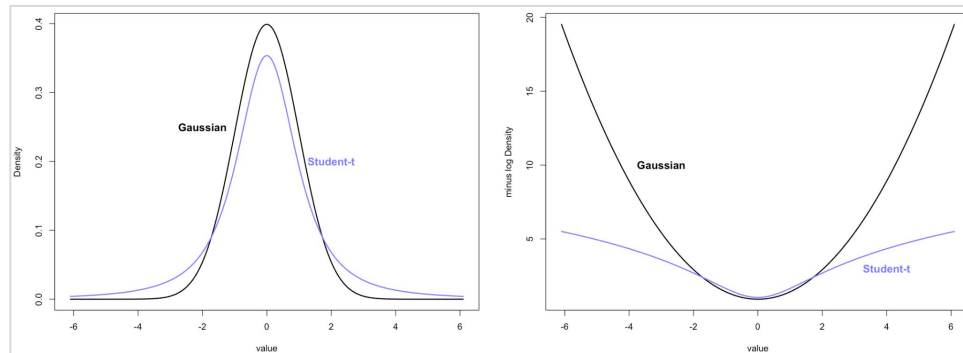
Model comparison: High pareto k values (outliers)? ...maybe switch to robust regression

The generalized Student-t distribution has the same mean μ and scale σ parameters as the Gaussian, but it also has an extra shape parameter ν that controls how thick the tails are:

- When ν is large, the tails are thin, converging in the limit $\nu = \infty$ to a Gaussian distribution
- But as ν approaches 1, the tails get thicker and rare extreme observations occur more often

Now, when we re-estimate the divorce model using a Student-t distribution with $\nu = 2$ and compute PSIS-LOO, **we do not get any warnings about Pareto k values**

The relative influence of Idaho has been much reduced



```
> PSIS(m5.3t)
Some Pareto k values are very high (>1). Set pointwise=TRUE to inspect individual points.
      PSIS      lppd  penalty std_err
1 130.5648 -65.28238  6.536672 15.91042
>
> PSIS(m5.3t)
      PSIS      lppd  penalty std_err
1 134.15 -67.075  7.106117 11.88921
>
> b5.3t <- brm(bf(D ~ 1 + M + A, nu = 2), family = student, data = d, refresh=0)
Compiling Stan program...
Start sampling
>
> ( loo_b5.3t <- loo(b5.3t) )

Computed from 4000 by 50 log-likelihood matrix

      Estimate      SE
elpd_loo    -67.0   5.8
p_loo         6.9   1.1
looic        134.1 11.5
-----
Monte Carlo SE of elpd_loo is 0.1.

All Pareto k estimates are good (k < 0.5).
See help('pareto-k-diagnostic') for details.
```

Appendix

Resources

[Regression and Other Stories](#)

[Statistical Rethinking](#)

[Statistical rethinking with brms, ggplot2, and the tidyverse: Second edition](#)

[Bayes Rules!](#)

[Tidy Modeling with R](#)

[Doing Bayesian Data Analysis, Second edition](#)

[Doing Bayesian Data Analysis in brms and the tidyverse](#)

[rstanarm vignettes](#)

[bayesplot vignettes](#)

[R for Data Science](#)

[R Graphics Cookbook](#)