Lecture 2

Diagnostics and Model Evaluation

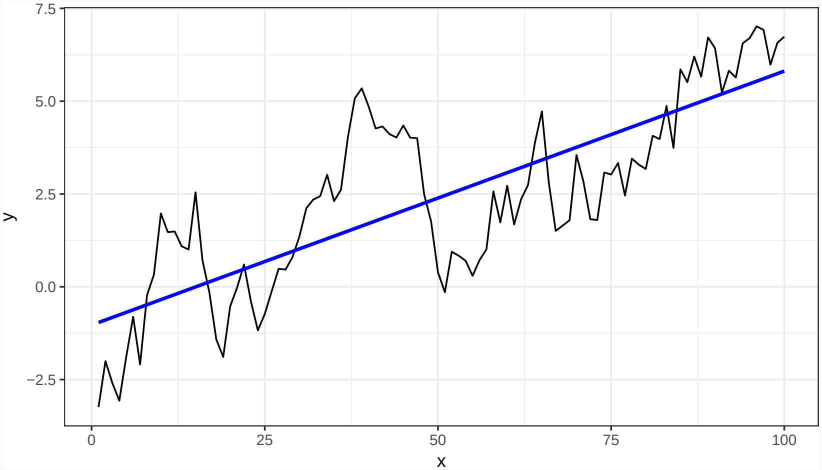
Colin Rundel

1/23/2018

Some more linear models

Linear model and data

```r
ggplot(d, aes(x=x,y=y)) +
  geom_line() +
  geom_smooth(method="lm", color="blue", se = FALSE)
```



Linear model

```
l = lm(y ~ x, data=d)

summary(l)
##
## Call:
## lm(formula = y ~ x, data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.6041 -1.2142 -0.1973  1.1969  3.7072
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.030315   0.310326   -3.32  0.00126 **
## x            0.068409   0.005335   12.82  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.54 on 98 degrees of freedom
## Multiple R-squared:  0.6266, Adjusted R-squared:  0.6227
## F-statistic: 164.4 on 1 and 98 DF,  p-value: < 2.2e-16
```

Bayesian model specification (JAGS)

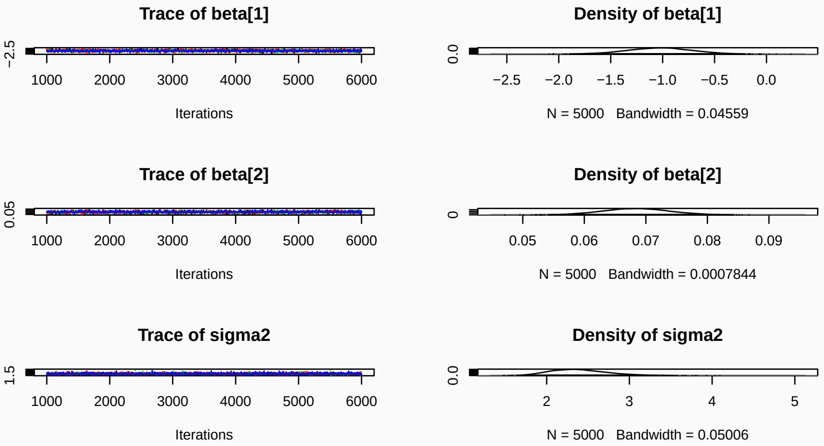
```
model =
"model{
  # Likelihood
  for(i in 1:length(y)){
    y[i] ~ dnorm(mu[i], tau)
    mu[i] = beta[1] + beta[2]*x[i]
  }

  # Prior for beta
  for(j in 1:2){
    beta[j] ~ dnorm(0,1/100)
  }

  # Prior for sigma / tau2
  tau ~ dgamma(1, 1)
  sigma2 = 1/tau
}"
```

Bayesian model fitting (JAGS)

```
n_burn = 1000; n_iter = 5000

m = rjags::jags.model(
  textConnection(model), data=d,
  quiet=TRUE, n.chains = 4
)
update(m, n.iter=n_burn, progress.bar="none")
samp = rjags::coda.samples(
  m, variable.names=c("beta","sigma2"),
  n.iter=n_iter, progress.bar="none"
)

str(samp, max.level=1)
## List of 4
##  $ : mcmc [1:5000, 1:3] -1.051 -1.154 -1.363 -0.961 -0.775 ...
##   ..- attr(*, "dimnames")=List of 2
##   ..- attr(*, "mcpar")= num [1:3] 1001 6000 1
##  $ : mcmc [1:5000, 1:3] -0.602 -0.175 -0.397 -0.555 -0.54 ...
##   ..- attr(*, "dimnames")=List of 2
##   ..- attr(*, "mcpar")= num [1:3] 1001 6000 1
##  $ : mcmc [1:5000, 1:3] -0.927 -1.5 -1.591 -1.726 -1.445 ...
##   ..- attr(*, "dimnames")=List of 2
##   ..- attr(*, "mcpar")= num [1:3] 1001 6000 1
##  $ : mcmc [1:5000, 1:3] -1.161 -1.179 -1.089 -1.099 -0.927 ...
##   ..- attr(*, "dimnames")=List of 2
##   ..- attr(*, "mcpar")= num [1:3] 1001 6000 1
##  - attr(*, "class")= chr "mcmc.list"
```

coda

```r
plot(samp)
```

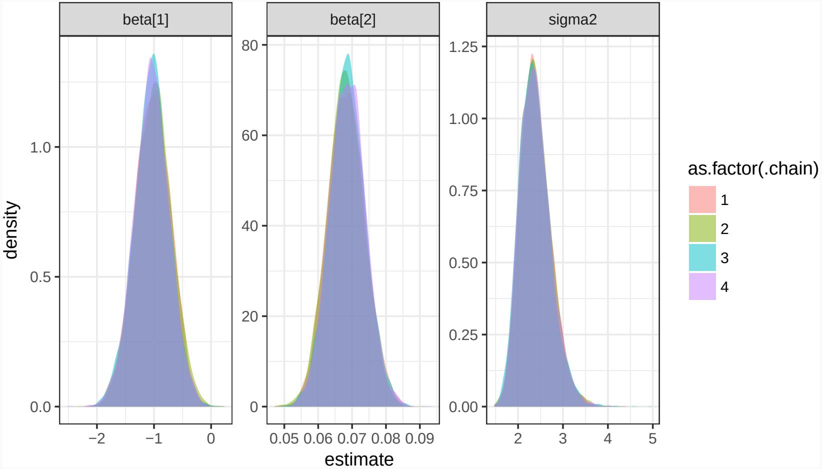


tidybayes

```
df_mcmc = tidybayes::gather_samples(samp, beta[i], sigma2) %>%
  mutate(parameter = paste0(term, ifelse(is.na(i),"",paste0("[",i,"]")))) %>%
  group_by(parameter, .chain)

head(df_mcmc)
## # A tibble: 6 x 6
## # Groups: parameter, .chain [2]
##   .chain .iteration     i term  estimate parameter
##    <int>      <int> <int> <chr>    <dbl> <chr>
## 1      1          1     1 beta   -1.05   beta[1]
## 2      1          1     2 beta    0.0645 beta[2]
## 3      1          2     1 beta   -1.15   beta[1]
## 4      1          2     2 beta    0.0726 beta[2]
## 5      1          3     1 beta   -1.36   beta[1]
## 6      1          3     2 beta    0.0713 beta[2]

tail(df_mcmc)
## # A tibble: 6 x 6
## # Groups: parameter, .chain [1]
##   .chain .iteration     i term   estimate parameter
##    <int>      <int> <int> <chr>     <dbl> <chr>
## 1      4       4995    NA sigma2     1.67 sigma2
## 2      4       4996    NA sigma2     2.68 sigma2
## 3      4       4997    NA sigma2     2.58 sigma2
## 4      4       4998    NA sigma2     1.93 sigma2
## 5      4       4999    NA sigma2     2.05 sigma2
## 6      4       5000    NA sigma2     2.00 sigma2
```

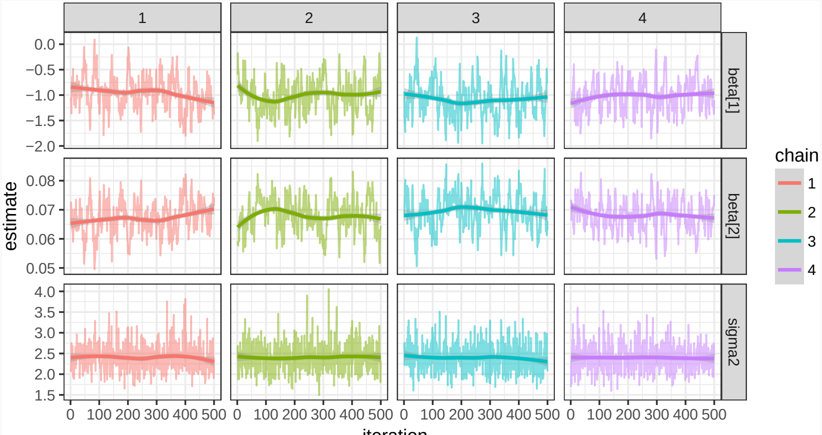
Posterior plots

```
ggplot(df_mcmc,aes(fill=as.factor(.chain), group=.chain, x=estimate)) +
  geom_density(alpha=0.5, color=NA) +
  facet_wrap(~ parameter,scales = "free")
```



Trace plots

```
df_mcmc %>% filter(.iteration <= 500) %>%
ggplot(aes(x=.iteration, y=estimate, color=as.factor(.chain))) +
  geom_line(alpha=0.5) +
  facet_grid(parameter~.chain, scale="free_y") +
  geom_smooth(method="loess") + labs(color="chain")
```



Credible Intervals

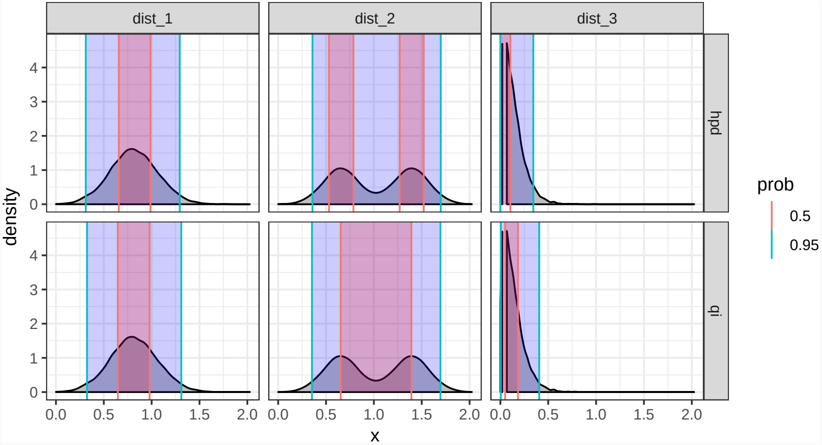
```
df_ci = tidybayes::mean_hdi(df_mcmc, estimate, .prob=c(0.8, 0.95))

df_ci
## # A tibble: 24 x 6
## # Groups: parameter, .chain [12]
##    parameter .chain estimate conf.low conf.high .prob
##    <chr>      <int>    <dbl>    <dbl>     <dbl> <dbl>
##  1 beta[1]        1  -1.02    -1.40     -0.587  0.800
##  2 beta[1]        2  -1.01    -1.44     -0.634  0.800
##  3 beta[1]        3  -1.04    -1.44     -0.656  0.800
##  4 beta[1]        4  -1.04    -1.44     -0.656  0.800
##  5 beta[2]        1   0.0683   0.0613    0.0755 0.800
##  6 beta[2]        2   0.0681   0.0613    0.0752 0.800
##  7 beta[2]        3   0.0686   0.0621    0.0753 0.800
##  8 beta[2]        4   0.0687   0.0621    0.0755 0.800
##  9 sigma2         1   2.40     1.96      2.81   0.800
## 10 sigma2         2   2.40     1.95      2.80   0.800
## # ... with 14 more rows
```

Aside - `mean_qi` vs `mean_hdi`

These differ in the use of the quantile interval vs. the highest-density interval.

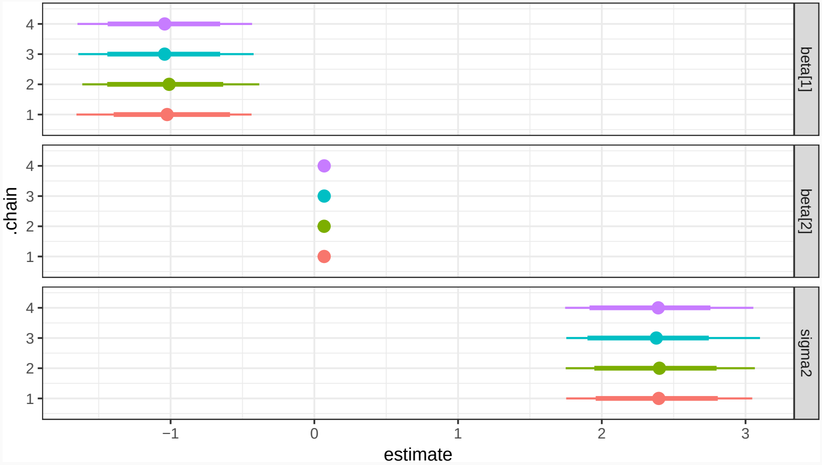
Aside - `mean_qi` vs `mean_hdi`

These differ in the use of the quantile interval vs. the highest-density interval.



Caterpillar Plots

```r
df_ci %>%
  ggplot(aes(x=estimate, y=.chain, color=as.factor(.chain))) +
  facet_grid(parameter~.) +
  tidybayes::geom_pointintervalh() +
  ylim(0.5,4.5)
```



Prediction

Revised model

```
model_pred =
"model{
  # Likelihood
  for(i in 1:length(y)){
    mu[i] = beta[1] + beta[2]*x[i]
    y[i] ~ dnorm(mu[i], tau)
    y_pred[i] ~ dnorm(mu[i], tau)
  }

  # Prior for beta
  for(j in 1:2){
    beta[j] ~ dnorm(0,1/100)
  }

  # Prior for sigma / tau2
  tau ~ dgamma(1, 1)
  sigma2 = 1/tau
}"
```

Revised fitting

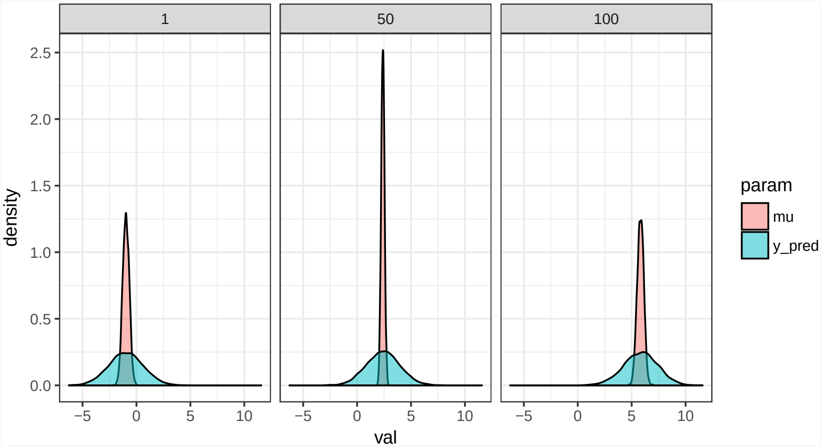
```r
n_burn = 1000; n_iter = 5000

m = rjags::jags.model(
  textConnection(model_pred), data=d,
  quiet=TRUE, n.chains = 1
)

update(m, n.iter=n_burn, progress.bar="none")

pred = rjags::coda.samples(
  m, variable.names=c("beta","sigma2","mu","y_pred","y","x"),
  n.iter=n_iter, progress.bar="none"
)
```

Predictions

```
df_pred = tidybayes::spread_samples(pred, y_pred[i], y[i], x[i], mu[i]) %>%
  mutate(resid = y - mu)

df_pred
## # A tibble: 500,000 x 8
## # Groups: i [100]
##    .chain .iteration     i y_pred      y     x       mu   resid
##     <int>      <int> <int>  <dbl>  <dbl> <dbl>   <dbl>   <dbl>
##  1      1          1     1 -0.858 -3.24   1.00 -0.554  -2.69
##  2      1          1     2 -0.638 -2.00   2.00 -0.496  -1.51
##  3      1          1     3  0.340 -2.59   3.00 -0.438  -2.16
##  4      1          1     4 -2.69  -3.07   4.00 -0.380  -2.69
##  5      1          1     5 -1.29  -1.88   5.00 -0.322  -1.56
##  6      1          1     6  0.758 -0.807  6.00 -0.264  -0.543
##  7      1          1     7  1.93  -2.09   7.00 -0.206  -1.89
##  8      1          1     8  3.00  -0.227  8.00 -0.148  -0.0794
##  9      1          1     9 -1.20   0.333  9.00 -0.0900  0.423
## 10      1          1    10 -0.515  1.98  10.0  -0.0320  2.01
## # ... with 499,990 more rows
```

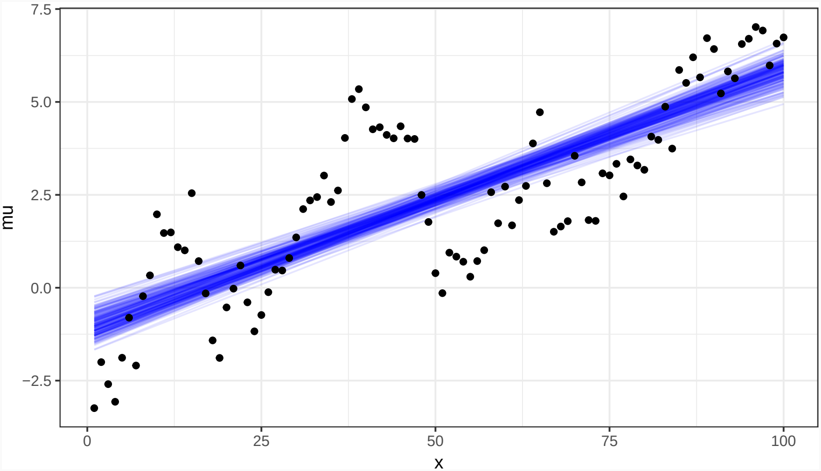
μ vs $y_{pred}

```
df_pred %>% ungroup() %>% filter(i %in% c(1,50,100)) %>% select(i, mu, y_pre
  ggplot(aes(x=val, fill=param)) + geom_density(alpha=0.5) + facet_wrap(~i)
```



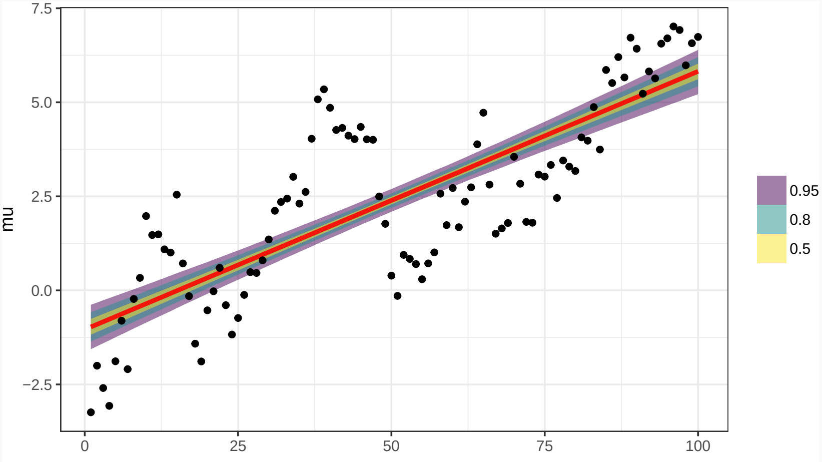
Predictions

```
df_pred %>% ungroup() %>% filter(.iteration <= 200) %>%
ggplot(aes(x=x)) +
  geom_line(aes(y=mu, group=.iteration), color="blue", alpha=0.1) +
  geom_point(data=d, aes(y=y))
```



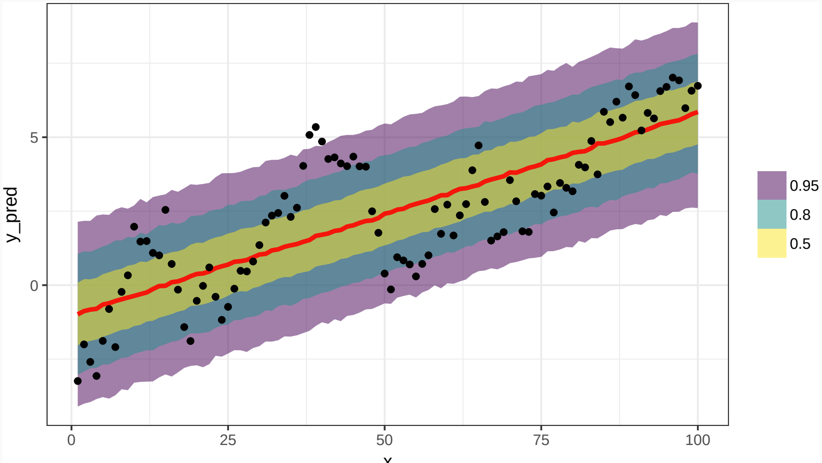
Posterior distribution (μ)

```
df_pred %>% ungroup() %>%
ggplot(aes(x=x)) +
  tidybayes::stat_lineribbon(aes(y=mu), alpha=0.5) +
  geom_point(data=d, aes(y=y))
```



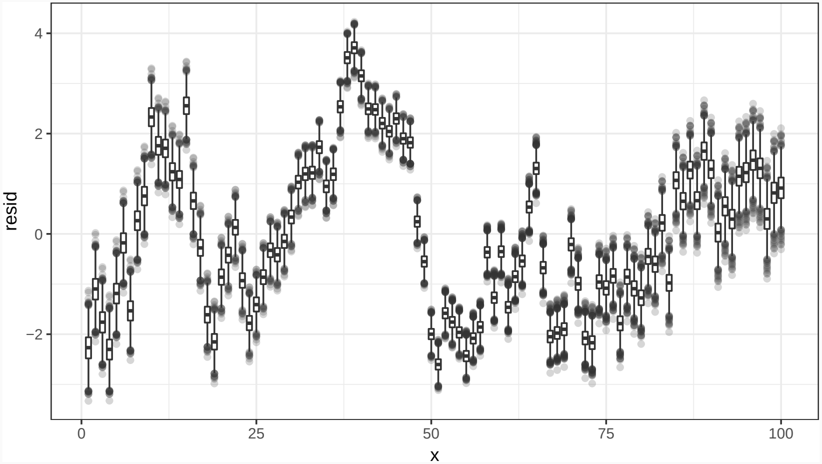
Posterior predictive distribution (y_{pred})

```
df_pred %>% ungroup() %>%
ggplot(aes(x=x)) +
  tidybayes::stat_lineribbon(aes(y=y_pred), alpha=0.5) +
  geom_point(data=d, aes(y=y))
```



Residual plot

```r
df_pred %>% ungroup() %>%
  ggplot(aes(x=x, y=resid)) +
  geom_boxplot(aes(group=x), outlier.alpha = 0.2)
```



Model Evaluation

Model assessment?

If we think back to our first regression class, one common option is R^2 which gives us the variability in Y explained by our model.

Quick review:

Model assessment?

If we think back to our first regression class, one common option is R^2 which gives us the variability in Y explained by our model.

Quick review:

$$\underbrace{\sum_{i=1}^{n}\left(Y_i-\bar{Y}\right)^2}_{\text{Total}}=\underbrace{\sum_{i=1}^{n}\left(\hat{Y}_i-\bar{Y}\right)^2}_{\text{Model}}+\underbrace{\sum_{i=1}^{n}\left(Y_i-\hat{Y}_i\right)^2}_{\text{Error}}$$

Model assessment?

If we think back to our first regression class, one common option is R^2 which gives us the variability in Y explained by our model.

Quick review:

$$\underbrace{\sum_{i=1}^{n}\left(Y_i-\bar{Y}\right)^2}_{\text{Total}}=\underbrace{\sum_{i=1}^{n}\left(\hat{Y}_i-\bar{Y}\right)^2}_{\text{Model}}+\underbrace{\sum_{i=1}^{n}\left(Y_i-\hat{Y}_i\right)^2}_{\text{Error}}$$

$$R^2=\frac{\sum_{i=1}^{n}\left(\hat{Y}_i-\bar{Y}\right)^2}{\sum_{i=1}^{n}\left(Y_i-\bar{Y}\right)^2}=\text{Cor}(\mathbf{Y},\hat{\mathbf{Y}})^2=\frac{\text{Var}(\hat{\mathbf{Y}})}{\text{Var}(\mathbf{Y})}$$

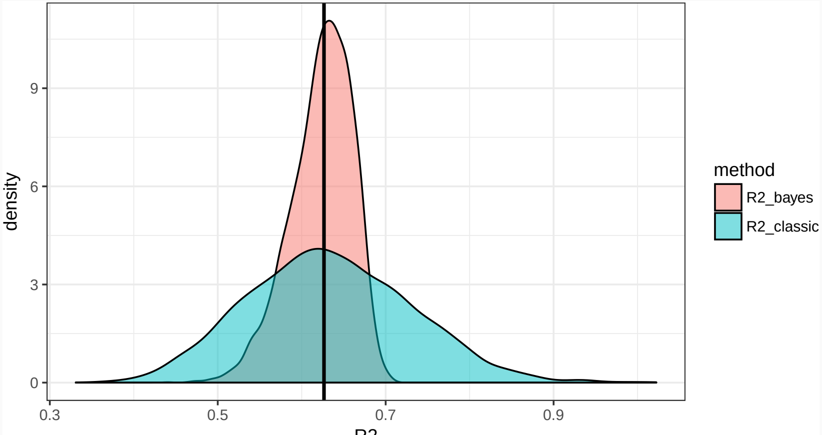
Bayesian R^2

When we compute any statistic for our model we want to do so at each iteration so that we can obtain the posterior distribution of that particular statistic (e.g. the posterior distribution of R^2 in this case).

```
df_R2 = df_pred %>%
  group_by(.iteration) %>%
  summarize(
    R2_classic = var(mu) / var(y),
    R2_bayes   = var(mu) / (var(mu) + var(resid))
  )

df_R2
## # A tibble: 5,000 x 3
##    .iteration R2_classic R2_bayes
##         <int>      <dbl>    <dbl>
##  1          1      0.450    0.537
##  2          2      0.448    0.536
##  3          3      0.462    0.545
##  4          4      0.511    0.574
##  5          5      0.583    0.609
##  6          6      0.470    0.550
##  7          7      0.529    0.584
##  8          8      0.451    0.538
```

Uh oh ...

```
df_R2 %>%
  tidyr::gather(method, R2, -.iteration) %>%
  ggplot(aes(x=R2, fill=method)) +
    geom_density(alpha=0.5) +
    geom_vline(xintercept=summary(l)$r.squared, size=1)
```



What if we collapsed first?

```
df_pred %>%
  group_by(i) %>%
  summarize(mu = mean(mu), y=mean(y), resid=mean(resid)) %>%
  summarize(
    R2_classic = var(mu) / var(y),
    R2_bayes   = var(mu) / (var(mu) + var(resid))
  )
## # A tibble: 1 x 2
##   R2_classic R2_bayes
##        <dbl>    <dbl>
## 1      0.630    0.628

summary(l)$r.squared
## [1] 0.6265565
```

Some problems with R^2

Some new issues,

- R^2 doesn't really make sense in the Bayesian context
    - multiple possible definitions with different properties
    - fundamental equality doesn't hold anymore

Some problems with R^2

Some new issues,

- R^2 doesn't really make sense in the Bayesian context
    - multiple possible definitions with different properties
    - fundamental equality doesn't hold anymore

Some old issues,

- R^2 always increases (or stays the same) when a predictor is added
- R^2 is highly susceptible to over fitting
- R^2 is sensitive to outliers
- R^2 depends heavily on current values of Y
- R^2 can differ drastically for two equivalent models (i.e. nearly identical inferences about key parameters)

Some Other Metrics

Root Mean Square Error

The traditional definition of rmse is as follows

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}$$

Root Mean Square Error

The traditional definition of rmse is as follows

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}$$

In the bayesian context, we have posterior samples from each parameter / prediction of interest so we can express this as

$$\text{RMSE} = \sqrt{\frac{1}{m}\frac{1}{n}\sum_{s=1}^{m}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i^s\right)^2}$$

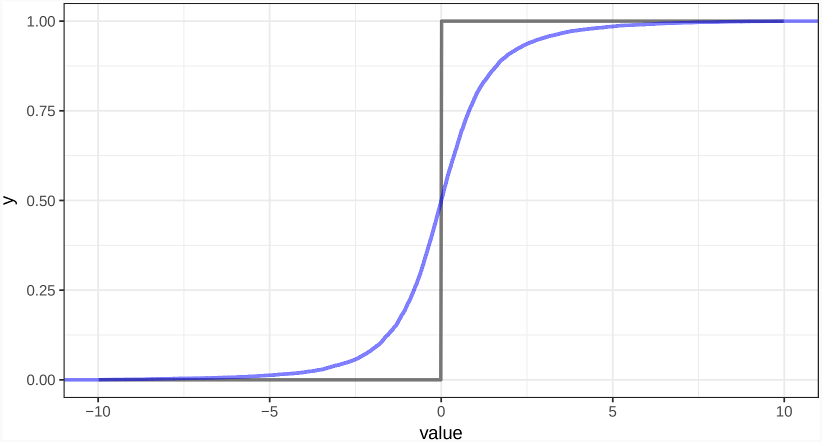
Continuous Rank Probability Score

Another approach is the continuous rank probability score which comes from the probabilistic forecasting literature, it compares the full posterior predictive distribution to the observation / truth.

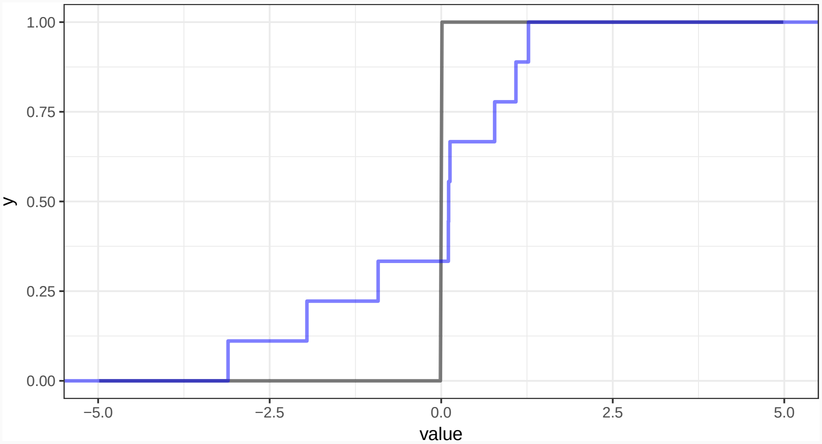
$$\text{CRPS} = \int_{-\infty}^{\infty} \left(F_{\hat{Y}}(z) - 1_{z \geq Y}\right)^2 dz$$

where $F_{\hat{Y}}$ is thes CDF of $\hat{Y}$ (the posterior predictive distribution for Y) and $1_{z \geq Y}$ is the indicator function which equals 1 when $z \geq Y$, the true/observed value of Y.

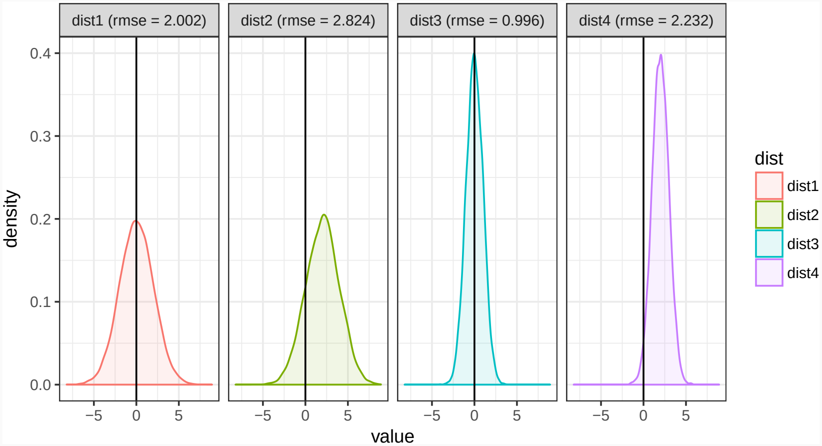
CDF vs Indicator



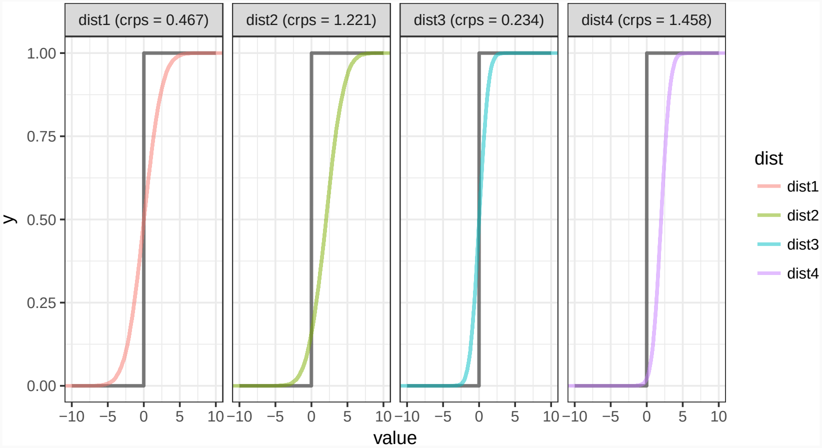
Empirical CDF vs Indicator



Accuracy vs. Precision - RMSE



Accuracy vs. Precision - CRPS



Empirical Coverage

One final method, which assesses model calibration is to examine how well credible intervals, derived from the posterior predictive distributions of the Ys, capture the true/observed values.

Empirical Coverage

One final method, which assesses model calibration is to examine how well credible intervals, derived from the posterior predictive distributions of the Ys, capture the true/observed values.

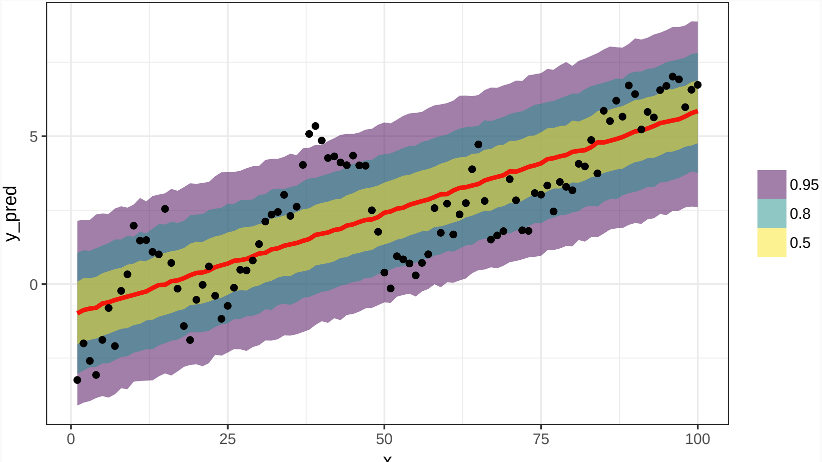
```
df_ec = df_pred %>%
  group_by(x,y) %>%
  tidybayes::mean_hdi(y_pred, .prob = c(0.5,0.8,0.9,0.95))

df_ec
## # A tibble: 400 x 6
## # Groups: x, y [100]
##         x      y y_pred conf.low conf.high .prob
##     <dbl>  <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1  1.00 -3.24  -0.972    -2.01      0.102 0.500
##  2  2.00 -2.00  -0.877    -1.60      0.513 0.500
##  3  3.00 -2.59  -0.845    -1.92      0.160 0.500
##  4  4.00 -3.07  -0.782    -1.77      0.299 0.500
##  5  5.00 -1.88  -0.679    -1.67      0.425 0.500
##  6  6.00 -0.807 -0.635    -1.67      0.439 0.500
##  7  7.00 -2.09  -0.543    -1.61      0.479 0.500
##  8  8.00 -0.227 -0.481    -1.64      0.390 0.500
##  9  9.00  0.333 -0.420    -1.61      0.505 0.500
## 10 10.0   1.98  -0.343    -1.37      0.708 0.500
## # ... with 390 more rows
```

Calculating Empirical Coverage

```
df_ec %>%
  mutate(contains = y >= conf.low & y <= conf.high) %>%
  group_by(prob=.prob) %>%
  summarize(emp_cov = sum(contains)/n())
## # A tibble: 4 x 2
##    prob emp_cov
##   <dbl>   <dbl>
## 1 0.500   0.400
## 2 0.800   0.770
## 3 0.900   0.950
## 4 0.950   0.970
```

Posterior predictive distribution (y_{pred})

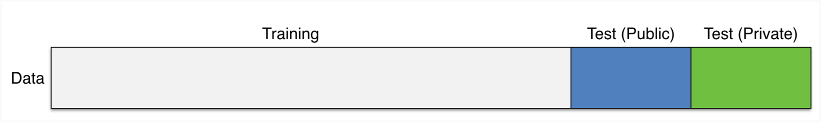
```r
df_pred %>% ungroup() %>%
ggplot(aes(x=x)) +
  tidybayes::stat_lineribbon(aes(y=y_pred), alpha=0.5) +
  geom_point(data=d, aes(y=y))
```



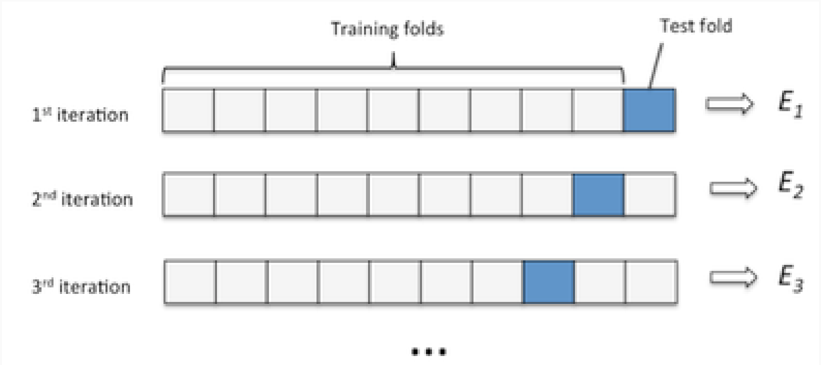
Cross-validation

Cross-validation styles

Kaggle style:



k-fold:



Cross-validation with `modelr`

```
d_kaggle = modelr::resample_partition(d, c(train=0.70, test1=0.15, test2=0.1
d_kaggle
## $train
## <resample [69 x 2]> 1, 3, 4, 6, 7, 8, 9, 10, 12, 13, ...
##
## $test1
## <resample [15 x 2]> 2, 11, 16, 19, 25, 29, 54, 57, 62, 69, ...
##
## $test2
## <resample [16 x 2]> 5, 15, 21, 23, 27, 32, 33, 36, 46, 47, ...

d_kfold = modelr::crossv_kfold(d, k=5)
d_kfold
## # A tibble: 5 x 3
##   train          test           .id
##   <list>         <list>         <chr>
## 1 <S3: resample> <S3: resample> 1
## 2 <S3: resample> <S3: resample> 2
## 3 <S3: resample> <S3: resample> 3
## 4 <S3: resample> <S3: resample> 4
## 5 <S3: resample> <S3: resample> 5
```

`resample` objects

The simple idea behind `resample` objects is that there is no need to create and hold on to these subsets / partitions of the original data frame - you only need to track which rows belong to what subset and then handle the creation of the new data frame when absolutely necessary.

```
d_kaggle$test1
## <resample [15 x 2]> 2, 11, 16, 19, 25, 29, 54, 57, 62, 69, ...

str(d_kaggle$test1)
## List of 2
##  $ data:Classes 'tbl_df', 'tbl' and 'data.frame':    100 obs. of  2 varia
##   ..$ x: int [1:100] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ y: num [1:100] -3.24 -2 -2.59 -3.07 -1.88 ...
##  $ idx : int [1:15] 2 11 16 19 25 29 54 57 62 69 ...
##  - attr(*, "class")= chr "resample"

as.data.frame(d_kaggle$test1)
## # A tibble: 15 x 2
##        x      y
##    <int>  <dbl>
##  1     2 -2.00
##  2    11  1.47
```

Simple usage

Model:

```
lm_train = lm(y~x, data=d_kaggle$train)
```

Simple usage

Model:

```
lm_train = lm(y~x, data=d_kaggle$train)
```

R^2:

```
lm_train %>% summary() %>% purrr::pluck("r.squared")
## [1] 0.631601

modelr::rsquare(lm_train, d_kaggle$train)
## [1] 0.631601
```

Simple usage

Model:

```
lm_train = lm(y~x, data=d_kaggle$train)
```

R^2:

```
lm_train %>% summary() %>% purrr::pluck("r.squared")
## [1] 0.631601

modelr::rsquare(lm_train, d_kaggle$train)
## [1] 0.631601
```

RMSE:

```
y_hat_test1 = predict(lm_train, d_kaggle$test1)

(y_hat_test1 - as.data.frame(d_kaggle$test1)$y)^2 %>% mean() %>% sqrt()
## [1] 1.401952

modelr::rmse(lm_train, d_kaggle$test1)
## [1] 1.401952
```

Cross-validation in R with `modelr` + `purrr`

```
lm_models = purrr::map(d_kfold$train, ~ lm(y~x, data=.))
str(lm_models, max.level = 1)
## List of 5
##  $ 1:List of 12
##   ..- attr(*, "class")= chr "lm"
##  $ 2:List of 12
##   ..- attr(*, "class")= chr "lm"
##  $ 3:List of 12
##   ..- attr(*, "class")= chr "lm"
##  $ 4:List of 12
##   ..- attr(*, "class")= chr "lm"
##  $ 5:List of 12
##   ..- attr(*, "class")= chr "lm"

purrr::map2_dbl(lm_models, d_kfold$train, modelr::rsquare)
##         1         2         3         4         5
## 0.6027358 0.6108063 0.6434123 0.6380698 0.6324337

purrr::map2_dbl(lm_models, d_kfold$test, modelr::rmse)
##        1        2        3        4        5
## 1.582513 1.691084 1.411778 1.447585 1.535771
```