tidyposterior

Michael Rose

Intro

Bayesian analysis is used here to answer the question: "When looking at resampling results, are the differences between models 'real'?" To answer this, a model can be created where the outcome is the resampling statistics (e.g. accuracy or RMSE). These values are explained by the model types. In doing this, we can get parameter estimates for each model's effect on performance and make statistical and practical comparisons between models.

function summaries

perf_mod() - Bayesian analysis of resampling statistics

tidy(perf_mod) - Extract posterior distributions for models

contrast_models() - Estimate the difference between models

summary(posterior) - Summarize the posterior distributions of model statistics

summary(posterior_diff) - Summarize posterior differences of model differences

ggplot(posterior) - Visualize the posterior distributions of model statistics

ggplot(posterior_diff) - Visualize the posterior distributions of model differences

no_trans, logit_trans, Fisher_trans, ln_trans, inv_trans - simple transformations

Index Page

The tidyposterior package can be used to conduct post hoc analyses of resampling results generated by models.

For example, if two models are evaluated with the root mean squared error (RMSE) using 10-fold cross-validation, there are 10 paired statistics. These can be used to make comparisons between models without involving a test set.

Example 1

```
# load library
library(tidyposterior)
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------------------------------

## v ggplot2 3.1.0       v purrr   0.3.1
## v tibble  2.0.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
## -- Conflicts ----------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# attach data
data("precise_example")

# look at data
precise_example %>% head()
```

```
## # A tibble: 6 x 29
##   splits id    glm_Accuracy glm_Kappa glm_ROC glm_Sens glm_Spec glm_PRAUC
## * <lgl>  <chr>        <dbl>     <dbl>   <dbl>    <dbl>    <dbl>     <dbl>
## 1 NA     Fold~        0.722     0.328   0.798    0.729    0.720     0.489
## 2 NA     Fold~        0.696     0.290   0.778    0.720    0.691     0.456
## 3 NA     Fold~        0.701     0.297   0.790    0.723    0.696     0.486
## 4 NA     Fold~        0.704     0.316   0.795    0.763    0.691     0.497
## 5 NA     Fold~        0.721     0.324   0.797    0.722    0.721     0.481
## 6 NA     Fold~        0.711     0.303   0.780    0.706    0.712     0.484
## # ... with 21 more variables: glm_Precision <dbl>, glm_Recall <dbl>,
## #   glm_F <dbl>, knn_Accuracy <dbl>, knn_Kappa <dbl>, knn_ROC <dbl>,
## #   knn_Sens <dbl>, knn_Spec <dbl>, knn_PRAUC <dbl>, knn_Precision <dbl>,
## #   knn_Recall <dbl>, knn_F <dbl>, nnet_Accuracy <dbl>, nnet_Kappa <dbl>,
## #   nnet_ROC <dbl>, nnet_Sens <dbl>, nnet_Spec <dbl>, nnet_PRAUC <dbl>,
## #   nnet_Precision <dbl>, nnet_Recall <dbl>, nnet_F <dbl>
```

```
# get classification accuracy results for each cross validated fold
accuracy <- precise_example %>%
  select(id, contains("Accuracy")) %>%
  setNames(tolower(gsub("_Accuracy$", "", names(.))))

accuracy
```

```
## # A tibble: 10 x 4
##    id       glm   knn  nnet
##    <chr>  <dbl> <dbl> <dbl>
##  1 Fold01 0.722 0.478 0.783
##  2 Fold02 0.696 0.492 0.766
##  3 Fold03 0.701 0.459 0.781
##  4 Fold04 0.704 0.454 0.783
##  5 Fold05 0.721 0.468 0.789
##  6 Fold06 0.711 0.474 0.776
##  7 Fold07 0.702 0.489 0.775
##  8 Fold08 0.718 0.458 0.785
##  9 Fold09 0.720 0.508 0.786
## 10 Fold10 0.719 0.440 0.777
```

```
# model accuracy results
acc_model <- perf_mod(accuracy, seed = 13311, verbose = FALSE)
```

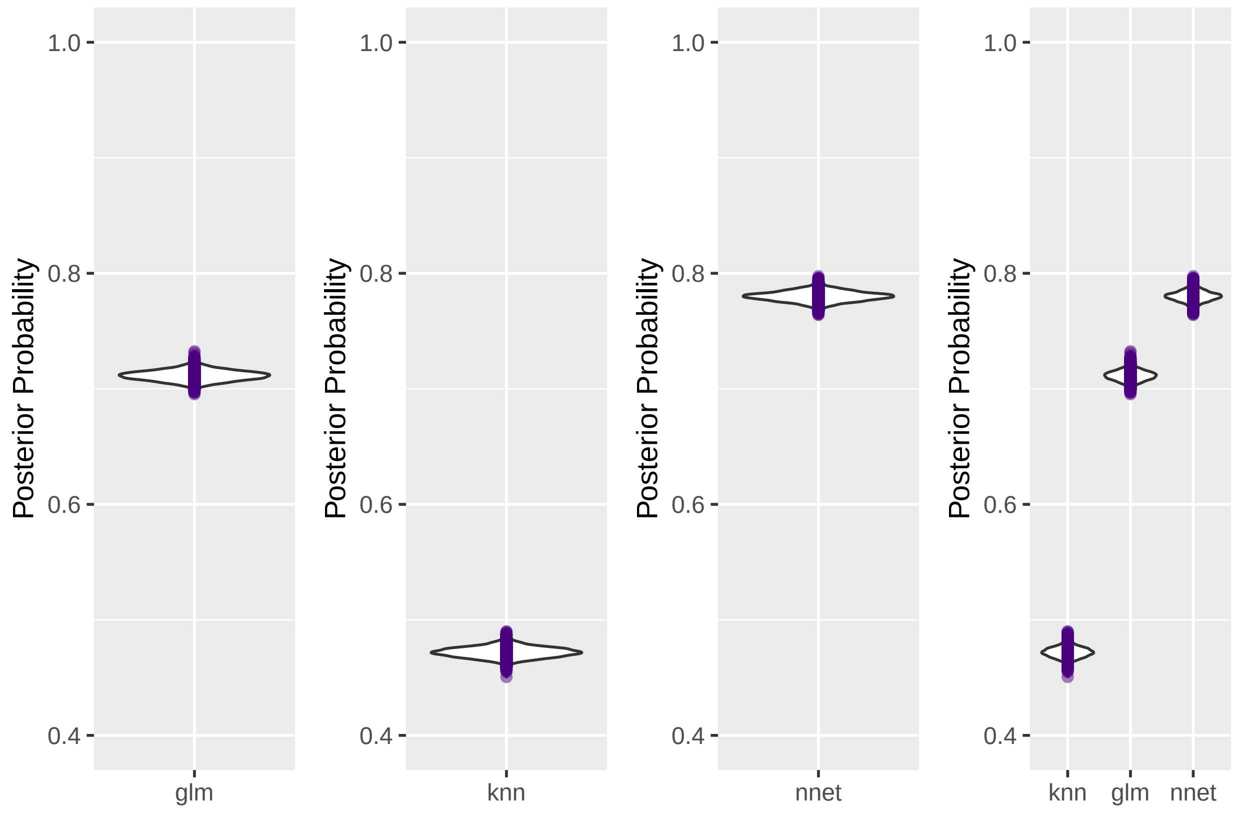
```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 9.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.98 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 2.49148 seconds (Warm-up)
## Chain 1:                0.441774 seconds (Sampling)
## Chain 1:                2.93325 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.44 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 2.62965 seconds (Warm-up)
## Chain 2:                0.438468 seconds (Sampling)
## Chain 2:                3.06811 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.4e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.34 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 2.46561 seconds (Warm-up)
## Chain 3:                0.3728 seconds (Sampling)
## Chain 3:                2.83841 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 3.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.34 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 2.72002 seconds (Warm-up)
## Chain 4:                0.271799 seconds (Sampling)
## Chain 4:                2.99182 seconds (Total)
## Chain 4:
```

```r
# extract posterior distributions
accuracy_dists <- tidy(acc_model)

# credible intervals for accuracy per model
summary(accuracy_dists)
```

```
## # A tibble: 3 x 4
##   model  mean lower upper
##   <chr> <dbl> <dbl> <dbl>
## 1 glm   0.711 0.704 0.719
## 2 knn   0.472 0.464 0.480
## 3 nnet  0.780 0.772 0.788
```

Plotting

```r
# create plot function
plot_model_posteriors <- function(data, model_chosen, geom_chosen = geom_point){
  data %>%
    filter(model %in% model_chosen) %>%
    ggplot(aes(x = model, y = posterior)) +
    geom_chosen(alpha = 0.5, col = "#4B0082") +
    scale_y_continuous(limits = c(0.4, 1))
}

# generate mapper
pmp <- as_mapper(partial(plot_model_posteriors, accuracy_dists))

# grab model list
model_list <- unique(accuracy_dists$model)

# map
posterior_plots <- map(model_list, pmp)

# create all together
all_together <- accuracy_dists %>% plot_model_posteriors(model_list)

gridExtra::grid.arrange(posterior_plots[[1]], posterior_plots[[2]], posterior_plots[[3]], all_together,
```



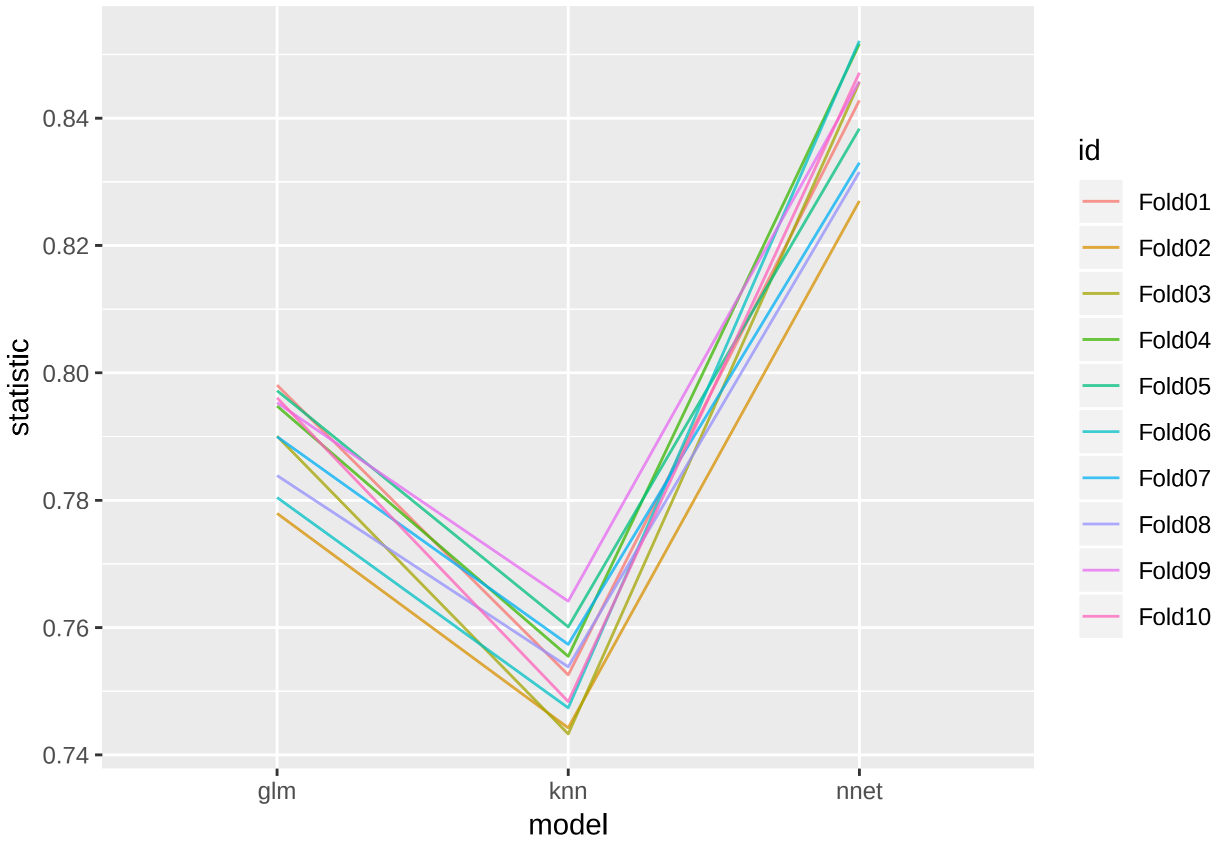
Example 2

The example that will be used is from the analysis of a fairly large classification data set using 10 fold cross validation with three models.

```
# check area under ROC curve
rocs <- precise_example %>%
  select(id, contains("ROC")) %>%
  set_names(tolower(gsub("_ROC$", "", names(.))))

rocs
```

```
## # A tibble: 10 x 4
##    id       glm   knn  nnet
##    <chr>  <dbl> <dbl> <dbl>
##  1 Fold01 0.798 0.753 0.843
##  2 Fold02 0.778 0.744 0.827
##  3 Fold03 0.790 0.743 0.846
##  4 Fold04 0.795 0.755 0.852
##  5 Fold05 0.797 0.760 0.838
##  6 Fold06 0.780 0.747 0.852
##  7 Fold07 0.790 0.757 0.833
##  8 Fold08 0.784 0.754 0.832
##  9 Fold09 0.795 0.764 0.846
## 10 Fold10 0.796 0.748 0.847
```

```r
# gather ROCs for plotting comparison
rocs_stacked <- gather(rocs, key = model, value = statistic, -id)

# plot
ggplot(rocs_stacked, aes(x = model, y = statistic, group = id, col = id)) +
  geom_line(alpha = 0.75)
```



Since the lines are fairly parallel, there is likely to be a strong resample-to-resample effect. Note that the variation is fairly small; the within-model results don't vary a lot and are not near the ceiling of performance (i.e. an AUC of one). It also seems pretty clear that the models are producing different levels of performance. There also seems to be roughly equal variation for each model despite the difference in performance.

A Basic Linear Model

When looking at resampling results, are the differences between models "real"?

To answer this, a model can be created where the outcome is the resampling statistics (area under the ROC curve). These values are explained by the model types. In doing this, we can get the parameter estimates for each model's effect on the resampled ROC values and make statistical and practical comparisons between models.

We will try a simple linear model with Gaussian errors that has a random effect for the resamples so that the within resample correlation can be estimated. Although the outcome is bounded in the interval [0, 1], the variability of these estimates might be precise enough to achieve a well fitting model.

To fit the model, perf_mod will be used to fit a model using the stan_glmer function in the rstanarm package:

```
# fit model
roc_model <- perf_mod(rocs, seed = 2824)
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.43 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 5.39224 seconds (Warm-up)
## Chain 1:                0.494009 seconds (Sampling)
## Chain 1:                5.88625 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4.8e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.48 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 4.49881 seconds (Warm-up)
## Chain 2:                0.401927 seconds (Sampling)
## Chain 2:                4.90074 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.5e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.35 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 5.54175 seconds (Warm-up)
## Chain 3:                0.437598 seconds (Sampling)
## Chain 3:                5.97935 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.4 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 4.34745 seconds (Warm-up)
## Chain 4:                0.267339 seconds (Sampling)
## Chain 4:                4.61479 seconds (Total)
## Chain 4:
```

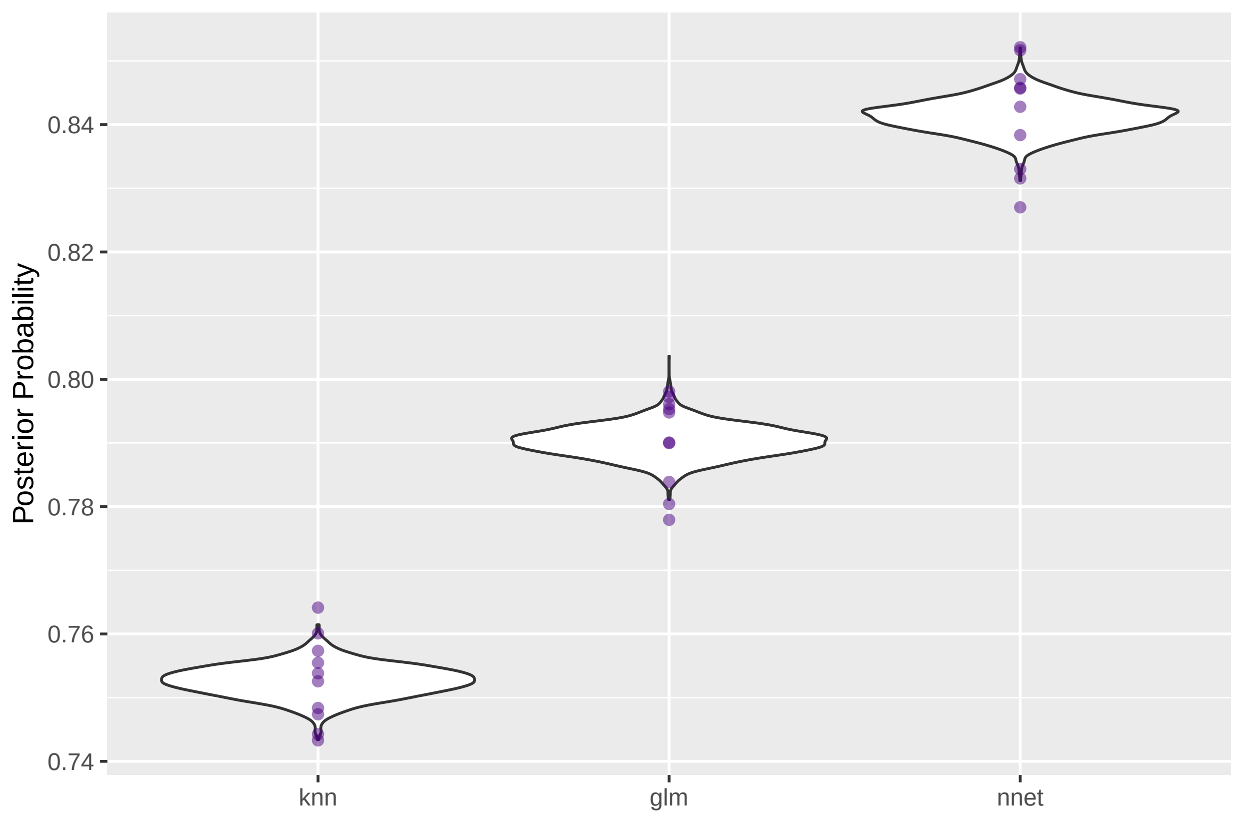
```
# The `stan_glmer` model is contained in the element `roc_model$stan`
roc_model$stan
```

```
## stan_glmer
##  family:       gaussian [identity]
##  formula:      statistic ~ model + (1 | id)
##  observations: 30
## ------
##             Median MAD_SD
## (Intercept) 0.8    0.0
## modelknn    0.0    0.0
## modelnnet   0.1    0.0
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 0.0    0.0
##
## Error terms:
##  Groups   Name        Std.Dev.
##  id       (Intercept) 0.0044
##  Residual             0.0069
## Num. levels: id 10
##
## Sample avg. posterior predictive distribution of y:
##          Median MAD_SD
## mean_PPD 0.8    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

To ensure the vailidity of this fit, the shinystan package can be used to generate an interactive assessment of the model results. One other thing that we can do is to examine the posterior distributions to see if they make sense in terms of the range of values.

Getting the Posterior Distributions

```
# the tidy function can be used to extract the distributions into a simple data frame
roc_post <- tidy(roc_model)

# plot
ggplot(roc_post) +
  # add the observed data to check for consistency
  geom_point(
    data = rocs_stacked,
    aes(x = model, y = statistic),
    alpha = 0.5, col = "#4B0082"
  )
```



These results look fairly reasonable given that we estimated a common variance for each of the models.

Comparing Models

We will compare the generalized linear model with the neural network. Before doing so, it helps to specify what a real difference between models would be. Suppose that a 2% increase in accuracy was considered to be substantial. We can add this into the analysis.

```
# compute the posterior for the difference in RMSE for the two models
glm_v_nnet <- contrast_models(roc_model, "nnet", "glm")

# look at data
head(glm_v_nnet)
```

```
##   difference model_1 model_2
## 1 0.05353329    nnet     glm
## 2 0.04916246    nnet     glm
## 3 0.05211164    nnet     glm
## 4 0.05317553    nnet     glm
## 5 0.05411325    nnet     glm
## 6 0.05102618    nnet     glm
```

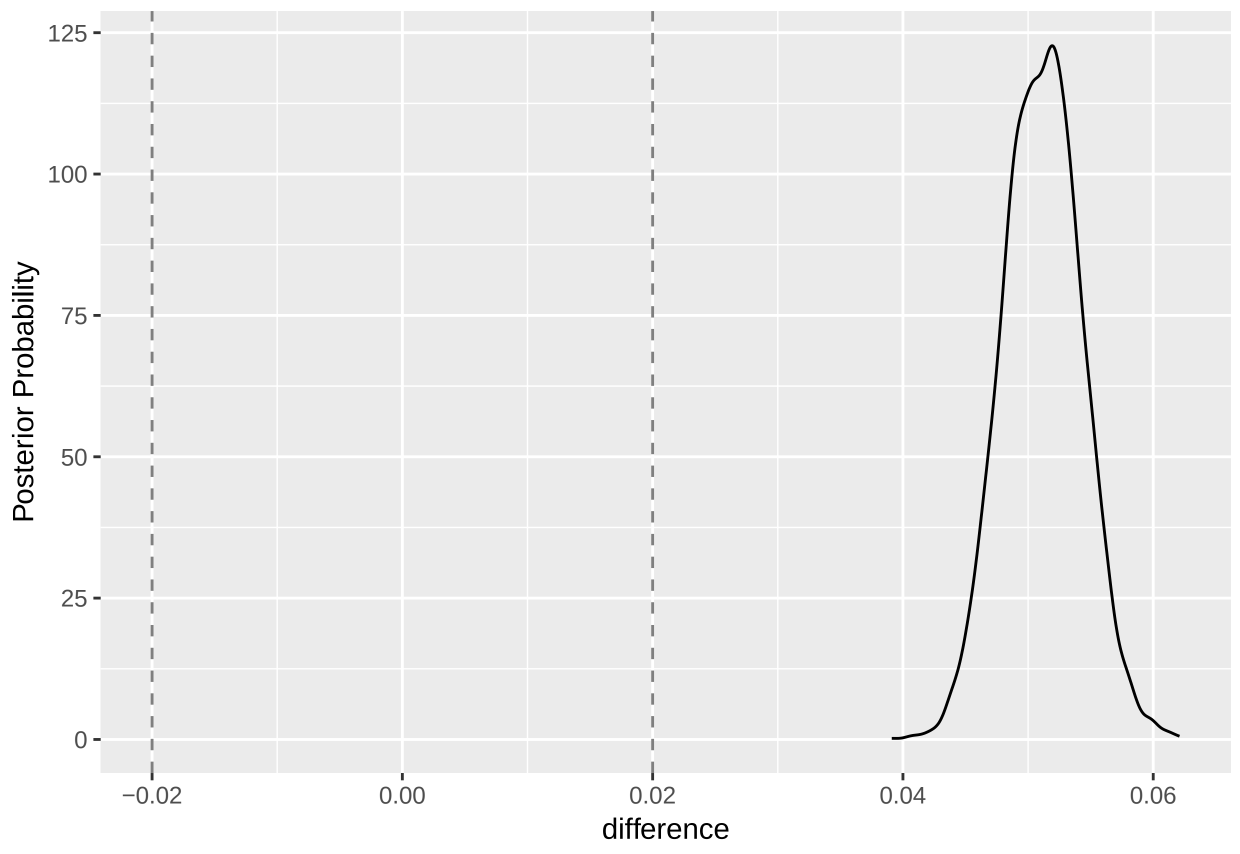
The summary function can be used to quantify this difference. It has an argument called size where we can add our belief about the size of a true difference.

```r
summary(glm_v_nnet, size = 0.02)
```

```
## # A tibble: 1 x 9
##   contrast probability   mean  lower  upper  size pract_neg pract_equiv
##   <chr>          <dbl>  <dbl>  <dbl>  <dbl> <dbl>     <dbl>       <dbl>
## 1 nnet vs~           1 0.0511 0.0461 0.0562  0.02         0           0
## # ... with 1 more variable: pract_pos <dbl>
```

The probability column indicates the proportion of the posterior distribution that is greater than 0. This result indicates that the entire distribution is larger than one. The credible intervals reflect the large difference in the area under the ROC curves for these models. The column pract_neg reflects the area where the posterior distribution is *less* than -2% (i.e. practically negative). Similarly, the pract_pos column shows that most of the area is greater than 2% which leads us to believe that this is truly a substantial difference in performance. The pract_equiv reflects how much of the posterior is in [-2%, 2%]. If this were near one, it might indicate that the models are not practically different based on the yardstick of 2%.

```r
# plot posterior of the differences
ggplot(glm_v_nnet, size = 0.02)
```



Example 3 | Different Bayesian Models

The dataset `noisy_example` contains the results of a series of regression models that were created from a small dataset with considerable variability. For resampling, 10 repeats of 10-fold cross validation were used to estimate performance. We will compare models using the root mean squared error.

```r
# load data
data("noisy_example")

# look at data
noisy_example %>% head()
```

```
## # A tibble: 6 x 15
##   splits id    id2   bag_RMSE bag_Rsquared bag_MAE cubist_RMSE
## * <lgl>  <chr> <chr>    <dbl>        <dbl>   <dbl>       <dbl>
## 1 NA     Repe~ Fold~     27.7       0.0205    22.6       14.0
## 2 NA     Repe~ Fold~     16.3       0.105     12.3        9.74
## 3 NA     Repe~ Fold~     21.6       0.224     18.2       12.7
## 4 NA     Repe~ Fold~     17.7       0.117     16.9       16.5
## 5 NA     Repe~ Fold~     14.0       0.101     12.6       19.0
## 6 NA     Repe~ Fold~     27.8       0.728     18.4       23.6
## # ... with 8 more variables: cubist_Rsquared <dbl>, cubist_MAE <dbl>,
## #   mars_RMSE <dbl>, mars_Rsquared <dbl>, mars_MAE <dbl>, nnet_RMSE <dbl>,
## #   nnet_Rsquared <dbl>, nnet_MAE <dbl>
```

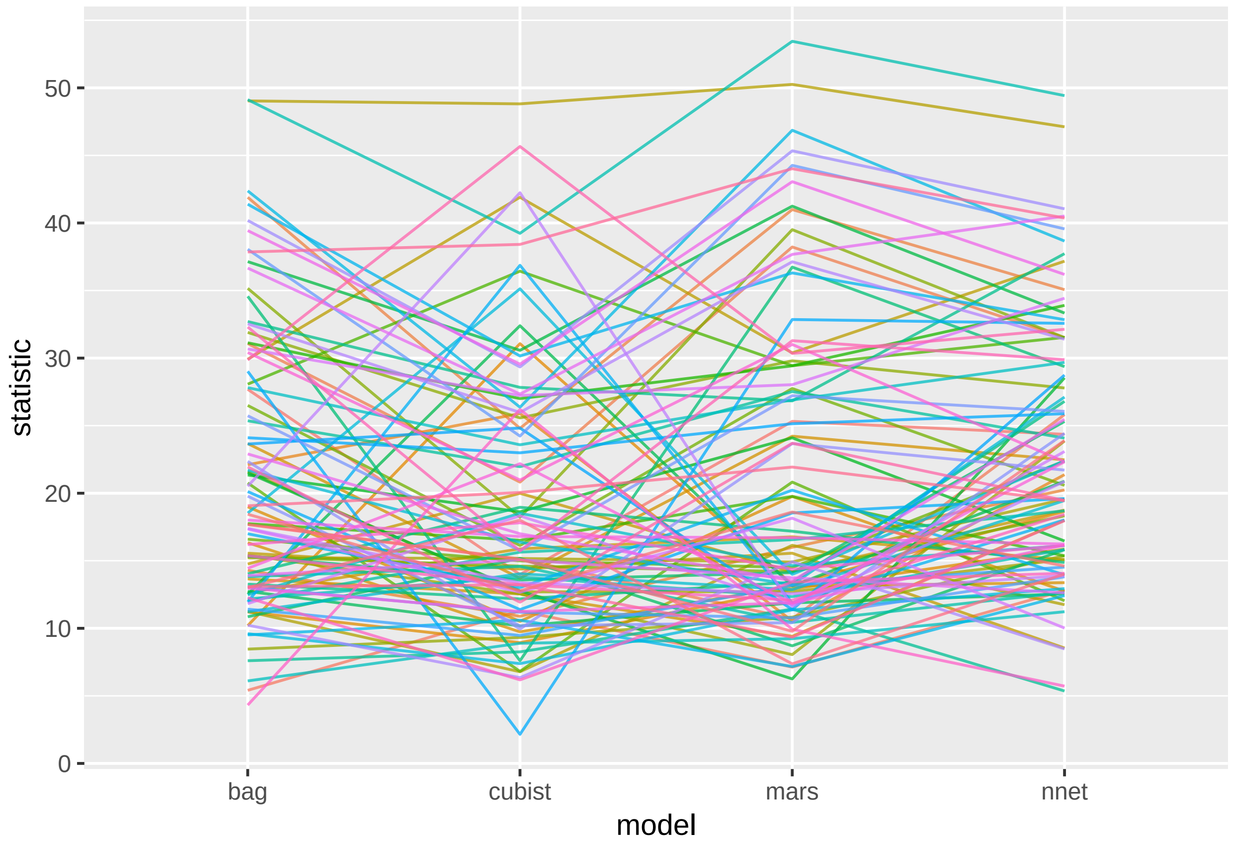
```r
# grab RMSEs
rmses <- noisy_example %>%
  select(id, id2, contains("RMSE")) %>%
  set_names(tolower(gsub("_RMSE$", "", names(.))))

rmses %>% head()
```

```
## # A tibble: 6 x 6
##   id       id2      bag cubist  mars  nnet
##   <chr>    <chr>  <dbl>  <dbl> <dbl> <dbl>
## 1 Repeat01 Fold01  27.7  14.0   25.3  24.4
## 2 Repeat02 Fold01  16.3   9.74  13.1  13.4
## 3 Repeat03 Fold01  21.6  12.7   12.7  15.1
## 4 Repeat04 Fold01  17.7  16.5   19.8  15.4
## 5 Repeat05 Fold01  14.0  19.0   17.2  14.9
## 6 Repeat06 Fold01  27.8  23.6   26.9  29.7
```

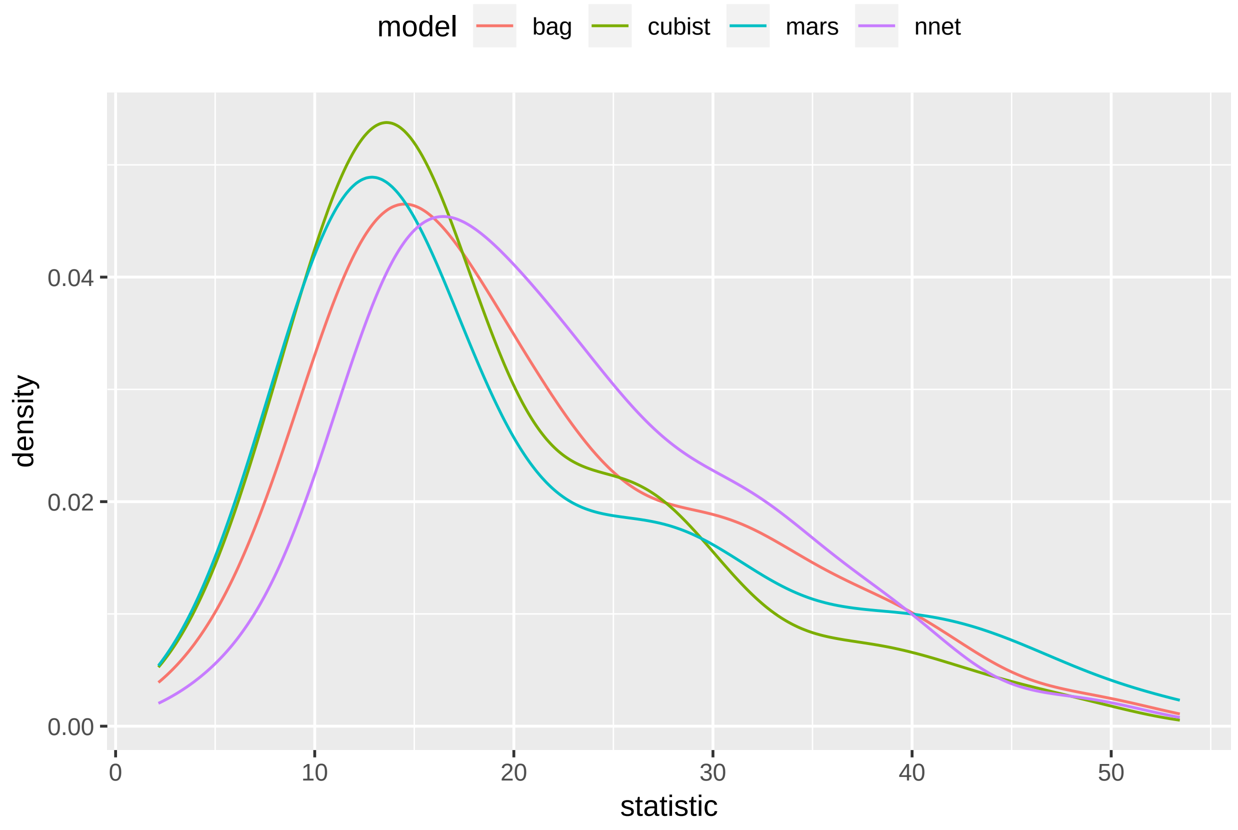
```r
# gather RMSE for comparison
stacked_rmse <- gather(rmses, key = model, value = statistic, -c(id, id2))

# summarize
mean_rmse <- stacked_rmse %>%
  group_by(model) %>%
  summarise(statistic = mean(statistic))

mean_rmse %>% head()
```

```
## # A tibble: 4 x 2
##   model  statistic
##   <chr>      <dbl>
## 1 bag         21.0
## 2 cubist      18.7
## 3 mars        20.5
## 4 nnet        22.3
```

```
# plot RMSE across different runs
ggplot(stacked_rmse, aes(
  x = model, y = statistic,
  group = paste(id, id2),
  col = paste(id, id2))) +
  geom_line(alpha = 0.75) +
  theme(legend.position = "none")
```



```
# plot RMSE densities across runs
ggplot(stacked_rmse, aes(col = model, x = statistic)) +
  geom_line(stat = "density", trim = FALSE) +
  theme(legend.position = "top")
```



A few observations: - The RMSE values vary 5-fold over the resampling results - Many of the lines cross, indicating that the resample-to-resample variability might be larger than the model-to-model variability - The violin plots show right skewed distributions that, given the variability, are approaching the asymptote of 0.

Transforming the Data

Another approach is to transform the RMSE values to something model symmetric and model the data on a different scale. A log transform will be used here using the built-in object ln_trans. In using this option, the posterior distributions are computed on the log scale and is automatically back-transformed into the original units. By not passing family to the function, we are using a Gaussian model.

```r
log_linear_model <- perf_mod(rmses, transform = ln_trans, seed = 74)
```

```
## Warning: Since no specific resampling method is known,the ID variables are
## collapsed into one column.
```

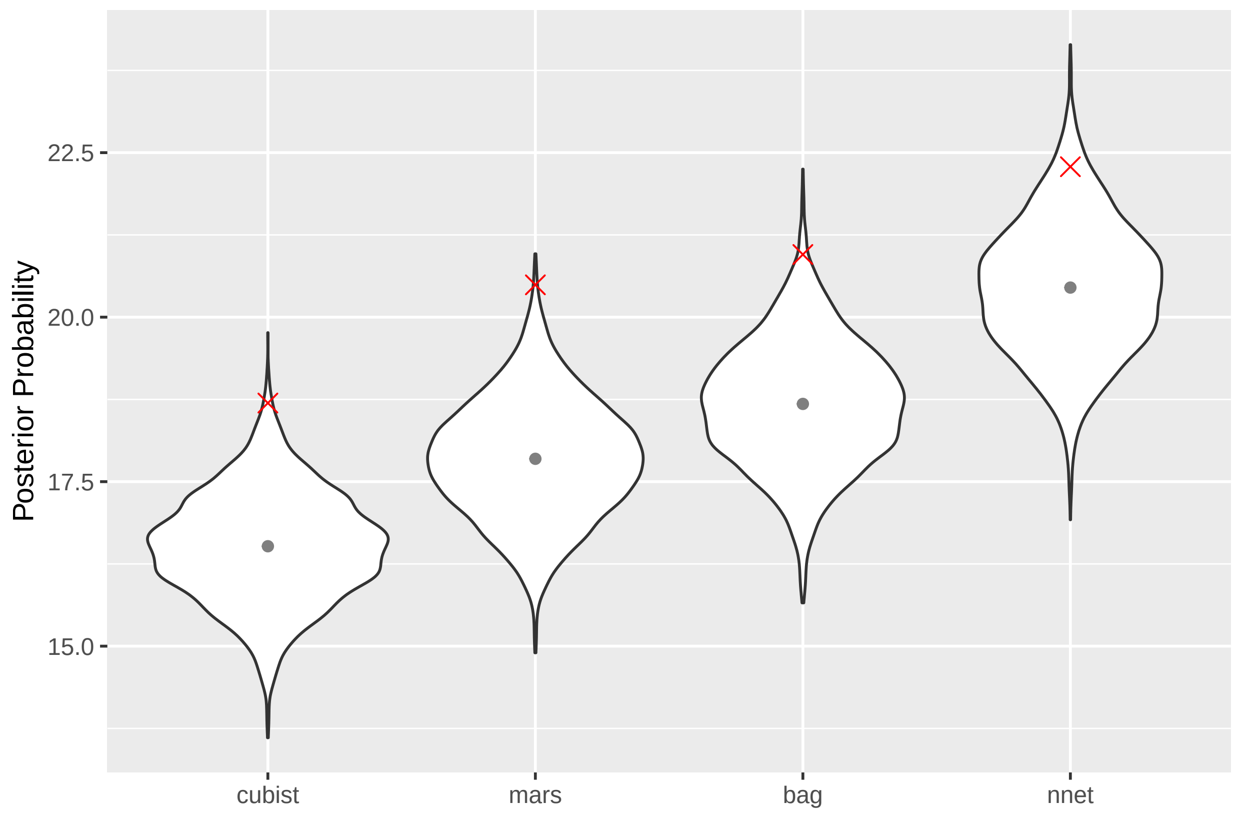
```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000143 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.43 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
```

```
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 2.66373 seconds (Warm-up)
## Chain 1:                2.25327 seconds (Sampling)
## Chain 1:                4.917 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000104 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.04 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 2.55331 seconds (Warm-up)
## Chain 2:                2.0928 seconds (Sampling)
## Chain 2:                4.64611 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000132 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.32 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 2.58457 seconds (Warm-up)
## Chain 3:                1.23366 seconds (Sampling)
## Chain 3:                3.81823 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000132 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.32 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 2.3981 seconds (Warm-up)
## Chain 4:                2.09126 seconds (Sampling)
## Chain 4:                4.48936 seconds (Total)
## Chain 4:
```

```r
# look at posterior and means
log_linear_post <- tidy(log_linear_model, seed = 3750)
log_linear_mean <- summary(log_linear_post)
log_linear_mean
```

```
## # A tibble: 4 x 4
##   model    mean lower upper
##   <chr>   <dbl> <dbl> <dbl>
## 1 bag      18.7  17.2  20.3
## 2 cubist   16.5  15.2  17.9
## 3 mars     17.8  16.4  19.4
## 4 nnet     20.4  18.8  22.2
```

```
# plot
ggplot(log_linear_post) +
  geom_point(data = log_linear_mean, aes(y = mean), alpha = 0.5) +
  geom_point(data = mean_rmse, aes(y = statistic),
             col = "red", pch = 4, cex = 3)
```



The posteriors are a lot less skewed by the observed and estimated means are still fairly far away from one another. Since these differences are in the same direction, this would not appear to be related to the shrinkage properties of Bayesian models.

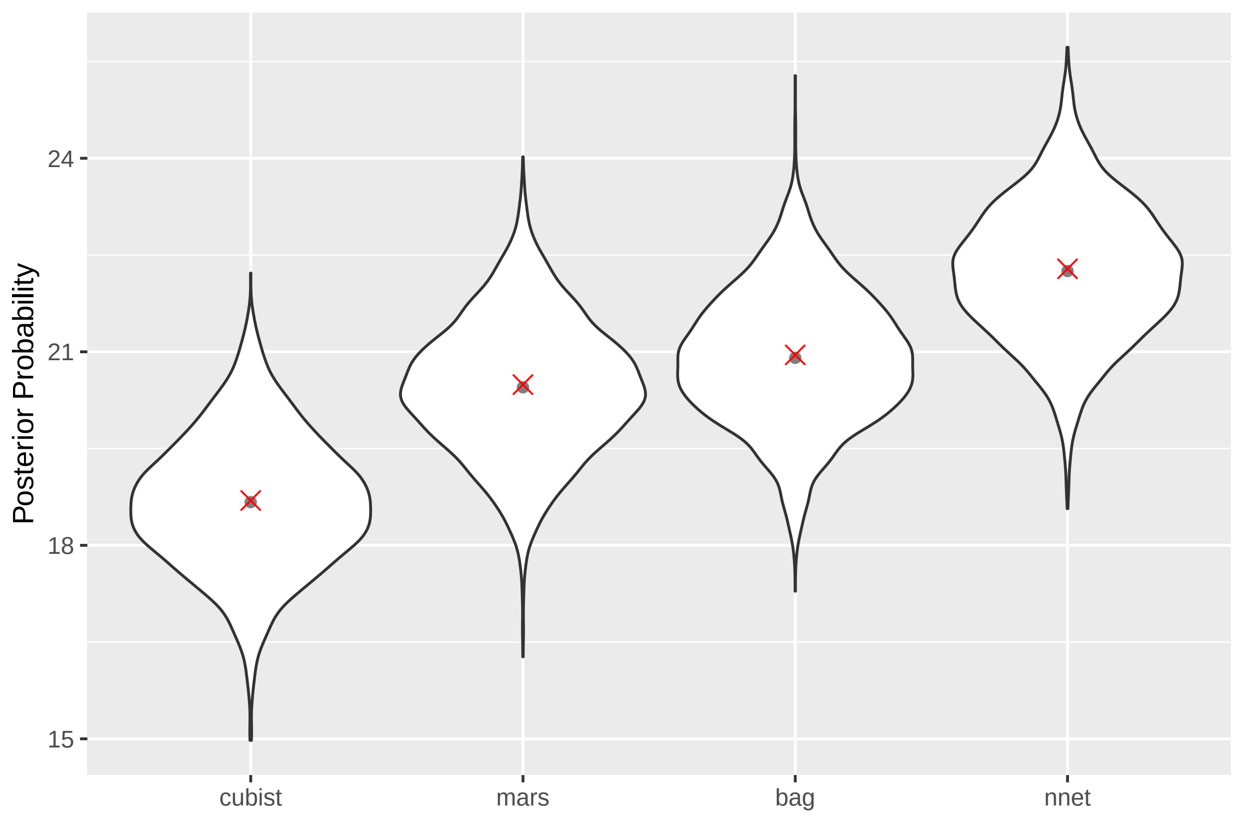
A Simple Gaussian Model

```
# fit a gaussian model for rmse estimates
linear_model <- perf_mod(rmses, seed = 74)
```

```
## Warning: Since no specific resampling method is known,the ID variables are
## collapsed into one column.
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000113 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.13 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 2.86573 seconds (Warm-up)
## Chain 1:                2.37211 seconds (Sampling)
## Chain 1:                5.23784 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000103 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.03 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 2.50409 seconds (Warm-up)
## Chain 2:                2.06199 seconds (Sampling)
## Chain 2:                4.56608 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000104 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.04 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 2.718 seconds (Warm-up)
## Chain 3:                2.18618 seconds (Sampling)
## Chain 3:                4.90418 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000142 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.42 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 2.40067 seconds (Warm-up)
## Chain 4:                2.19702 seconds (Sampling)
## Chain 4:                4.59768 seconds (Total)
## Chain 4:
```

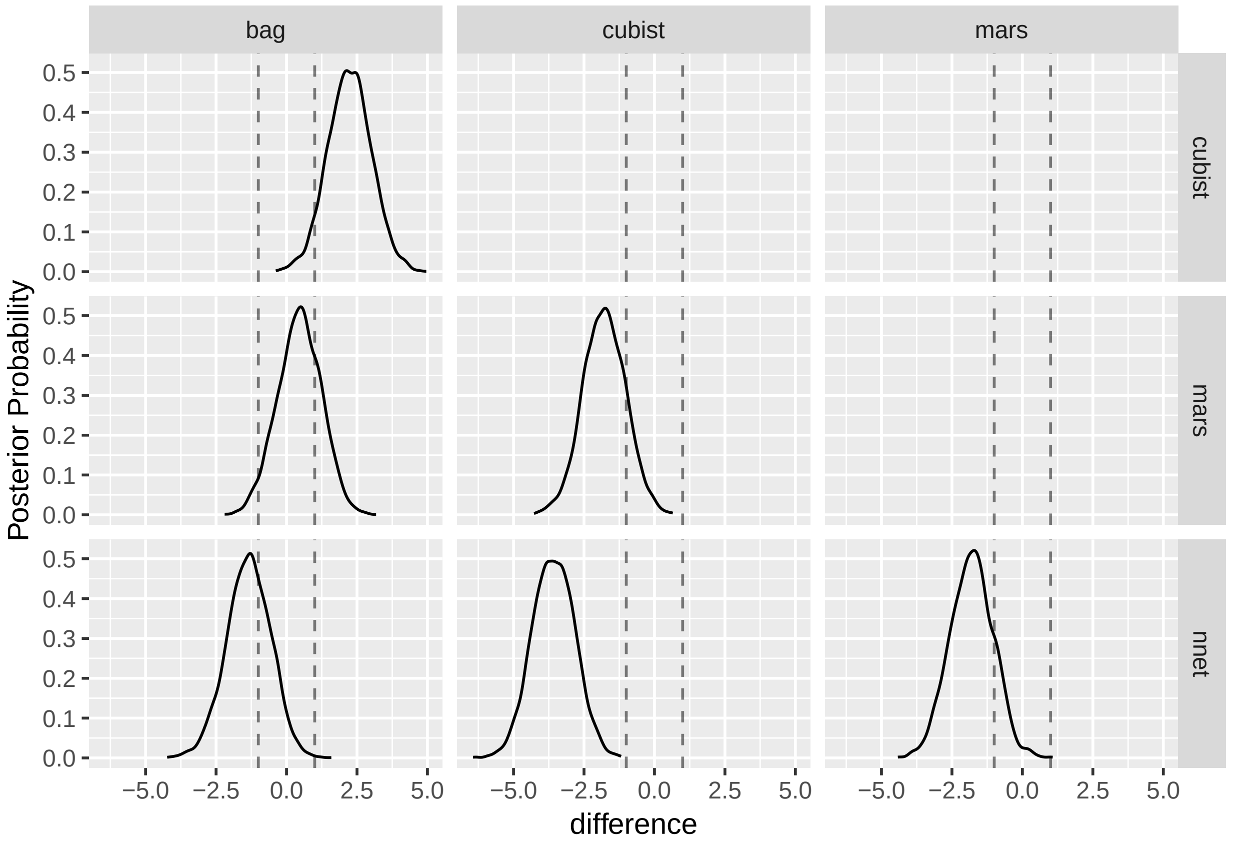
```r
# get posterior and derive means
linear_post <- tidy(linear_model, seed = 3750)
linear_mean <- summary(linear_post)

# plot
ggplot(linear_post) +
  geom_point(data = linear_mean, aes(y = mean), alpha = 0.5) +
  geom_point(data = mean_rmse, aes(y = statistic),
             col = "red", pch = 4, cex = 3)
```



These are right on target. Despite the skewness of the original data, a simple linear model did best here. In hindsight, this makes sense since we are modeling *summary statistics* as our outcome. Even if we believe these to be potentially skewed distributions, the central limit theorem is kicking in here and the estimates are trending to normality.

We can compare models using the contrast_models function. The function has arguments for two sets of models to compare but if these are left to their default (NULL), all pair wise combinations are used. Let's say that an RMSE difference of 1 unit is important.

```
# get contrasts
all_contrasts <- contrast_models(linear_model, seed = 8967)

# plot
ggplot(all_contrasts, size = 1)
```



```r
# get summary
summary(all_contrasts, size = 1)
```

```
## # A tibble: 6 x 9
##   contrast probability   mean  lower   upper  size pract_neg pract_equiv
##   <chr>          <dbl>  <dbl>  <dbl>   <dbl> <dbl>     <dbl>       <dbl>
## 1 bag vs ~     0.998    2.24   0.967  3.52       1    0           0.0555
## 2 bag vs ~     0.728    0.457 -0.823  1.71       1    0.0342      0.716
## 3 bag vs ~     0.0382  -1.35  -2.66  -0.0947     1    0.668       0.330
## 4 cubist ~     0.00925 -1.78  -3.02  -0.516      1    0.847       0.153
## 5 cubist ~     0       -3.58  -4.82  -2.35       1    1           0
## 6 mars vs~     0.0115  -1.80  -3.08  -0.577      1    0.846       0.154
## # ... with 1 more variable: pract_pos <dbl>
```

Based on our effect size of a single unit, the only pair that are practically equivalent are MARS and bagged trees. Since cubist has the smallest RMSE, it is not unreasonable to say that this model probides uniformly better results than the others shown here.

One Final Note

The Bayesian models have population parameters for the model effects (akin to "fixed" effects in mixed models) as well as variance parameter(s) related to the resamples. The posteriors computed only reflect the mean parameters and should only be used to make inferences about this data set generally. This posterior calculation could not be used to predict the level of performance for a model a new *resample* of the data. In this case, the variance paramaters come into play and the posterior would be much wider. In essence, the posteriors shown here are measuring the average performance value instead of a resample-specific value.