Bayesian data analysis on heart failure prediction

Jani Jokivuori, Arun Bhatia, Mai Hoang

Contents

1) Introduction & details about the dataset

Yearly almost 18 million people lose their lives because of cardiovascular diseases [1]. Cardiovascular diseases affect the heart, lungs and blood vessels and represent 32 percent of all global deaths. Due to this, it is crucial to notice cardiovascular disease early which helps us to manage and treat them. The dataset used in this project contains 299 observations with 13 clinical features for predicting death events related to cardiovascular diseases. Because cardiovascular deaths can be prevented with the correction of diet, an increase in physical inactivity and the elimination of alcohol and tobacco, it's important to detect cardiovascular diseases as early as possible [1]. People who are at the highest risk of getting a Cardiovascular disease need to have early detection. Our Bayesian models can be valuable in this issue. With the help of Bayesian data analysis, we might be able to detect cardiovascular diseases earlier and separate the ones that are at the highest risk of getting a cardiovascular disease. Below are some illustrative figure of the data.

The link to the dataset can be found in the references of this project which is located at the bottom of the report. Since the dataset is freely available on Kaggle and anyone can use it, there have been plenty of different types of analysis made on it. Some reports have tried to use machine learning with classification or decision trees. On the other hand, people have also tried to fit regression models into it. In total Kaggle has over 800 reports/codes that have used the dataset.

```
#Import the dataset
clinical_records <- read.csv(file = "heart_failure_clinical_records_dataset.csv")
clinical_data_for_stan_model<- list(observations = clinical_records,
                                    no_of_rows = nrow(clinical_records),
                                    no_of_cols = ncol(clinical_records))
cat("Number of rows (records) and columns (features and 1 death label):", nrow(clinical_records), ncol(
```

```
Number of rows (records) and columns (features and 1 death label): 299 13
```

Here is an example of the first record in the dataset

```
head(clinical_records, 1)
```

```
  age anaemia creatinine_phosphokinase diabetes ejection_fraction
1  75       0                     582        0                20
  high_blood_pressure platelets serum_creatinine serum_sodium sex smoking time
1                   1    265000              1.9          130   1       0    4
  DEATH_EVENT
1           1
```

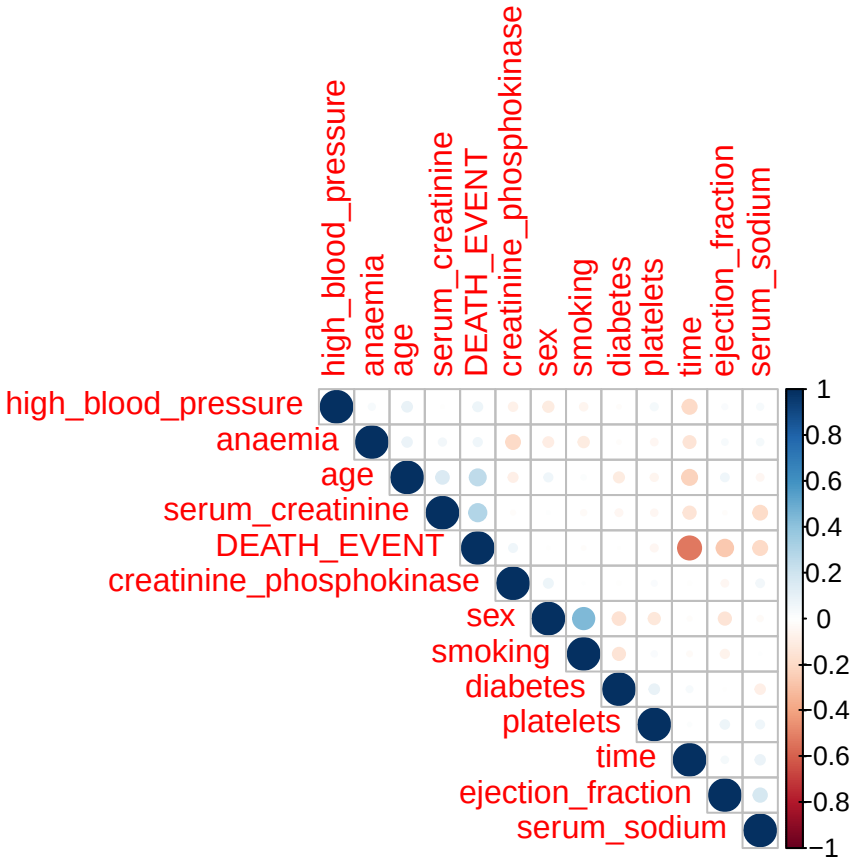
```
#Find rows contain binary values (True), or numerical values (False)
apply(clinical_records, 2, function(x) {all(x %in% 0:1)})
```

```
                      age                  anaemia creatinine_phosphokinase
                    FALSE                     TRUE                    FALSE
                 diabetes        ejection_fraction      high_blood_pressure
                     TRUE                    FALSE                     TRUE
                platelets         serum_creatinine             serum_sodium
                    FALSE                    FALSE                    FALSE
                      sex                  smoking                     time
                     TRUE                     TRUE                    FALSE
              DEATH_EVENT
                     TRUE
```

Out of 13 columns in the dataset, six are binary features, seven are numerical features, and one death_event row which denotes whether the patients die due to heart failure.

Below is a correlation plot of features. Red color indicates high negative correlation and blue color indicates high positive correlation. For example sex and smoking have a high correlation. On the other hand, death_event and time have high negative correlation.

```
corrplot(cor(clinical_records), type="upper", order="hclust")
```



2) Details about dataset

Definitions of the clinical features in the dataset: - Age: the patient's age.

- Anaemia: Whether the patient has a decrease of red blood cells or hemoglobin.
- Creatine phosphokinase is Level of the CPK enzyme in the blood (mcg/L). It is found mainly in the heart, brain, and skeletal muscle [2]. The average range is 10 to 120 mcg/L. Abnormal results indicates (filtered to related to heart only): heart attack, inflammation of the heart muscle.
- Diabetes: If the patient has diabetes
- Ejection fraction is the percentage of blood leaving the heart at each contraction (percentage) [3]. It reflects how sufficiently your heart pumps blood.

- Serum creatinine level is based on a blood test that measures the quantity of creatinine in your blood [4]. It describes how well a kidney is functioning. When your kidneys are not functioning well, your serum creatinine level goes up. In general, a normal level is: 0.7 - 1.3 mg/dL for males, 0.6 - 1.1 mg/dL for females.
- high_blood_pressure: If the patient has hypertension.
- platelets: Platelets in the blood (kiloplatelets/mL).
- serum_sodium: Level of serum sodium in the blood (mEq/L). Sodium help control the amount of fluid and the proportion of acids and bases (pH balance) in your body [5]. Sodium furthermore helps your nerves and muscles work appropriately.
- sex: male = 1, female = 0.
- smoking: whether the patient smoke.
- Time: is the number of days the patients is in the treatment, recorded when dead or left the treatment (recover)

3) Description of the two models used:

For this project, we use a linear and quadratic function in combination with logit transformation for modelling. The equations is shown as below:

$log(\frac{P(death|X)}{1-P(death|X)}) = w_1 * x_1 + w_2 * x_2 + ... + w_{12} * x_{12} + bias$

$P(death|X) = \frac{1}{1+e^{-(w_1*x_1+w_2*x_2+...w_{12}*x_{12}+bias)}}$

The observation is modeled as:

$y \sim B(P(death|X))$

We use bernoulli distribution, because it's a discrete distribution model for a binary outcome trial, in our case telling if the person is going to die or not [7]

In the quadratic model, we simply add a square term for each features, such that:

$log(\frac{P(death|X)}{1-P(death|X)}) = w_{11} * x_1 + w_{21} * x_1^2 + ... + bias$

4) Weakly informative priors

The model is supposed to estimates the weights w_{**} in the equations above. Since we do not know what is going to be the value for those parameters, we just make a wild guess by modelling each of the weights with a "wide enough" normal distributions of $N(0, 10)$. Thus this is our weakly informative priors.

```
#Weakly informative priors for the linear model
weights_prior_mean <- c(0,0,0,0,0,0,0,0,0,0,0,0)
weights_prior_std <- c(10,10,10,10,10,10,10,10,10,10,10,10)
bias_prior_mean <- 0
bias_prior_std <- 10
```

```r
#Weakly informative priors for the 2nd order model
weights_1st_order_prior_mean <- c(0,0,0,0,0,0,0,0,0,0,0,0)
weights_1st_prior_std <- c(10,10,10,10,10,10,10,10,10,10,10,10)
weights_2nd_order_prior_mean <- c(0,0,0,0,0,0,0,0,0,0,0,0)
weights_2nd_prior_std <- c(10,10,10,10,10,10,10,10,10,10,10,10)
bias_prior_mean <- 0
bias_prior_std <- 10
```

5) Stan models code

Here is the linear model used:

```r
cat(readLines('linear_model.stan'), sep = '\n')
```

```
data {
  int <lower=0> no_of_rows;
  int <lower=0> no_of_features;

  vector[no_of_features] features[no_of_rows];
  array[no_of_rows] int<lower=0> outcome;

  array[no_of_features] real weights_prior_mean;
  array[no_of_features] real<lower=0> weights_prior_std;

  real bias_prior_mean;
  real bias_prior_std;
}

//What we need to fits
parameters {
  vector[no_of_features] w;
  real bias;
}

model {
  bias ~ normal(bias_prior_mean, bias_prior_std);
  for (n in 1:no_of_features) {
      w[n] ~ normal(weights_prior_mean[n], weights_prior_std[n]);
  };

  vector[no_of_rows] linear_sum;
  for (n in 1:no_of_rows) {
      linear_sum[n] = 0;
      linear_sum[n] = dot_product(w, features[n,]);
      //add bias here
      linear_sum[n] += bias;
  };

  outcome ~ bernoulli_logit(linear_sum);
}
```

```
generated quantities {
  //y_pred =
  vector[no_of_features] priors;
  for (i in 1:no_of_features) {
      priors[i] = normal_rng(weights_prior_mean[i], weights_prior_std[i]);
  };
  real bias_prior = normal_rng(bias_prior_mean, bias_prior_std);

  vector[no_of_rows] log_lik;
  for (n in 1:no_of_rows) {
      real linear_sum_ = dot_product(w, features[n,]);
      linear_sum_ += bias;
      log_lik[n] =  bernoulli_logit_lpmf(outcome[n] | linear_sum_);
  };
}
```

Here is the 2nd order polynomial model used:

```
cat(readLines('non_linear_model.stan'), sep = '\n')
```

```
data {
  int <lower=0> no_of_rows;
  int <lower=0> no_of_features;

  vector[no_of_features] features[no_of_rows];
  array[no_of_rows] int<lower=0> outcome;

  array[no_of_features] real weights_1st_order_prior_mean;
  array[no_of_features] real<lower=0> weights_1st_prior_std;

  array[no_of_features] real weights_2nd_order_prior_mean;
  array[no_of_features] real<lower=0> weights_2nd_prior_std;

  real bias_prior_mean;
  real bias_prior_std;
}

//What we need to fits
parameters {
  vector[no_of_features] w1; //first order coeff
  vector[no_of_features] w2; //second order coeff
  real bias;
}

model {
  bias ~ normal(bias_prior_mean, bias_prior_std);
  for (n in 1:no_of_features) {
      w1[n] ~ normal(weights_1st_order_prior_mean[n], weights_1st_prior_std[n]);
      w2[n] ~ normal(weights_2nd_order_prior_mean[n], weights_2nd_prior_std[n]);
  };

  vector[no_of_rows] sum_;
```

```
  for (n in 1:no_of_rows) {
      //first order terms
      sum_[n] = dot_product(w1, features[n,]);
      //second order terms
      sum_[n] += dot_product(w2, square(features[n,]));
      //add bias here
      sum_[n] += bias;
      //print("og:", features[n,]);
      //print("square:", square(features[n,]));
  };
  //posterior
  outcome ~ bernoulli_logit(sum_);
}

generated quantities {
  //y_pred =
  vector[no_of_rows] log_lik;
  for (n in 1:no_of_rows) {
      real sum_ = dot_product(w1, features[n,]);
      sum_ += dot_product(w2, square(features[n,]));
      sum_ += bias;
      log_lik[n] =  bernoulli_logit_lpmf(outcome[n] | sum_);
  };
}
```

6) Stan options explanations

For the Stan model fitting, we runs a total of 4 chains, each with 10000 samples and another 10000 is discarded as warm up. We tried running 2000 iterations warm ups and 2000 samples but the models have not converged by then (even after we normalized non-binary features). The seed is set to 7 for result reproduction, and refresh is set to 0 to suppress the redundant print out in the report.

Another interesting point is that normalized non-binary features makes the model fitted much faster. If we use the raw data and fit our linear model with 20000 iterations (half as warm up), then it takes around 20 minutes. However after the normalization, the whole process takes about one minute. So the take away lesson is besides the fact that the features' range affect the final results, it also affects time for the model to reach the convergence as well. We still have this long time to convergence with the quadratic model as we just use the normalized features same as in the linear model without further preprocess, but due to time constraint we have not improve the preprocess for that particular model.

```r
#Preprocessing
#Normalize non binary features
preprocessed_clinical_records <- data.frame(clinical_records)
columns_to_preprocess <- c(1, 3, 5, 7, 8, 9, 11, 12)
preprocessed_clinical_records[columns_to_preprocess] <-
  as.data.frame(scale(preprocessed_clinical_records[columns_to_preprocess]))
```

```r
#Fit linear model
fit_linear_model <- function(weights_prior_mean,
                             weights_prior_std,
                             bias_prior_mean,
                             bias_prior_std){
  #print(preprocessed_clinical_records)
```

```
  #Prepare the data for the model
  clinical_data_for_stan_model<- list(no_of_rows = nrow(preprocessed_clinical_records),
                                      no_of_features = 12,
                                      features = preprocessed_clinical_records[, 1:12],
                                      outcome = preprocessed_clinical_records[, 13],
                                      weights_prior_mean = weights_prior_mean,
                                      weights_prior_std = weights_prior_std,
                                      bias_prior_mean = bias_prior_mean,
                                      bias_prior_std = bias_prior_std)
  #Fit the model
  stan_model <- cmdstan_model(stan_file = "linear_model.stan")
  stan_linear_model_fit <- stan_model$sample(data = clinical_data_for_stan_model,
                                             chains = 4,
                                             iter_warmup = 10000,
                                             iter_sampling = 10000,
                                             seed = 7,
                                             refresh=0)
  return (stan_linear_model_fit)
}
```

```
stan_linear_model_fit <- fit_linear_model(weights_prior_mean = weights_prior_mean,
                                      weights_prior_std = weights_prior_std,
                                      bias_prior_mean = bias_prior_mean,
                                      bias_prior_std = bias_prior_std)
```

```
Running MCMC with 4 parallel chains...

Chain 1 finished in 19.2 seconds.
Chain 2 finished in 19.2 seconds.
Chain 4 finished in 19.2 seconds.
Chain 3 finished in 19.5 seconds.

All 4 chains finished successfully.
Mean chain execution time: 19.3 seconds.
Total execution time: 19.7 seconds.
```

```
stan_model_summary <- stan_linear_model_fit$summary()[c(1,2,8,9,10)]
head(stan_model_summary, 5)
```

```
# A tibble: 5 x 5
  variable       mean  rhat ess_bulk ess_tail
  <chr>         <dbl> <dbl>    <dbl>    <dbl>
1 lp__     -117.       1.00   16344.   24098.
2 w[1]        0.609    1.00   59694.   30120.
3 w[2]       -0.00897  1.00   52949.   31792.
4 w[3]        0.260    1.00   64841.   28333.
5 w[4]        0.153    1.00   55490.   31090.
```

```
#Fit 2nd order polymonial model
fit_2nd_order_model <- function(weights_1st_order_prior_mean,
                                weights_1st_prior_std,
                                weights_2nd_order_prior_mean,
```

```
                                weights_2nd_prior_std,
                                bias_prior_mean,
                                bias_prior_std){
  #Prepare the data for the model
  clinical_data_for_stan_model<- list(no_of_rows = nrow(preprocessed_clinical_records),
                                      no_of_features = 12,
                                      features = preprocessed_clinical_records[, 1:12],
                                      outcome = preprocessed_clinical_records[, 13],
                                      weights_1st_order_prior_mean = weights_1st_order_prior_mean,
                                      weights_1st_prior_std = weights_1st_prior_std,
                                      weights_2nd_order_prior_mean = weights_2nd_order_prior_mean,
                                      weights_2nd_prior_std = weights_2nd_prior_std,
                                      bias_prior_mean = bias_prior_mean,
                                      bias_prior_std = bias_prior_std)
  #Fit the model
  stan_model <- cmdstan_model(stan_file = "non_linear_model.stan")
  stan_2nd_order_model_fit <- stan_model$sample(data = clinical_data_for_stan_model,
                                            chains = 4,
                                            iter_warmup = 10000,
                                            iter_sampling = 10000,
                                            seed = 7,
                                            refresh=0)
  return (stan_2nd_order_model_fit)
}
```

```
stan_model_2nd_order_fit <- 0
if(file.exists("stan_model_2nd_order_fit.RDS")) {
  stan_model_2nd_order_fit <- readRDS(file = "stan_model_2nd_order_fit.RDS")
} else {
  #Fit 2nd order model
  stan_model_2nd_order_fit <- fit_2nd_order_model(
                                    weights_1st_order_prior_mean = weights_1st_order_prior_mean,
                                    weights_1st_prior_std = weights_1st_prior_std,
                                    weights_2nd_order_prior_mean = weights_2nd_order_prior_mean,
                                    weights_2nd_prior_std = weights_2nd_prior_std,
                                    bias_prior_mean = bias_prior_mean,
                                    bias_prior_std = bias_prior_std)
  stan_model_2nd_order_fit$save_object(file = "stan_model_2nd_order_fit.RDS")
}

stan_model_2nd_order_draws <- stan_model_2nd_order_fit$draws()
```

```
stan_model_summary_2 <- stan_model_2nd_order_fit$summary()[c(1,2,8,9,10)]
head(stan_model_summary_2, 5)
```

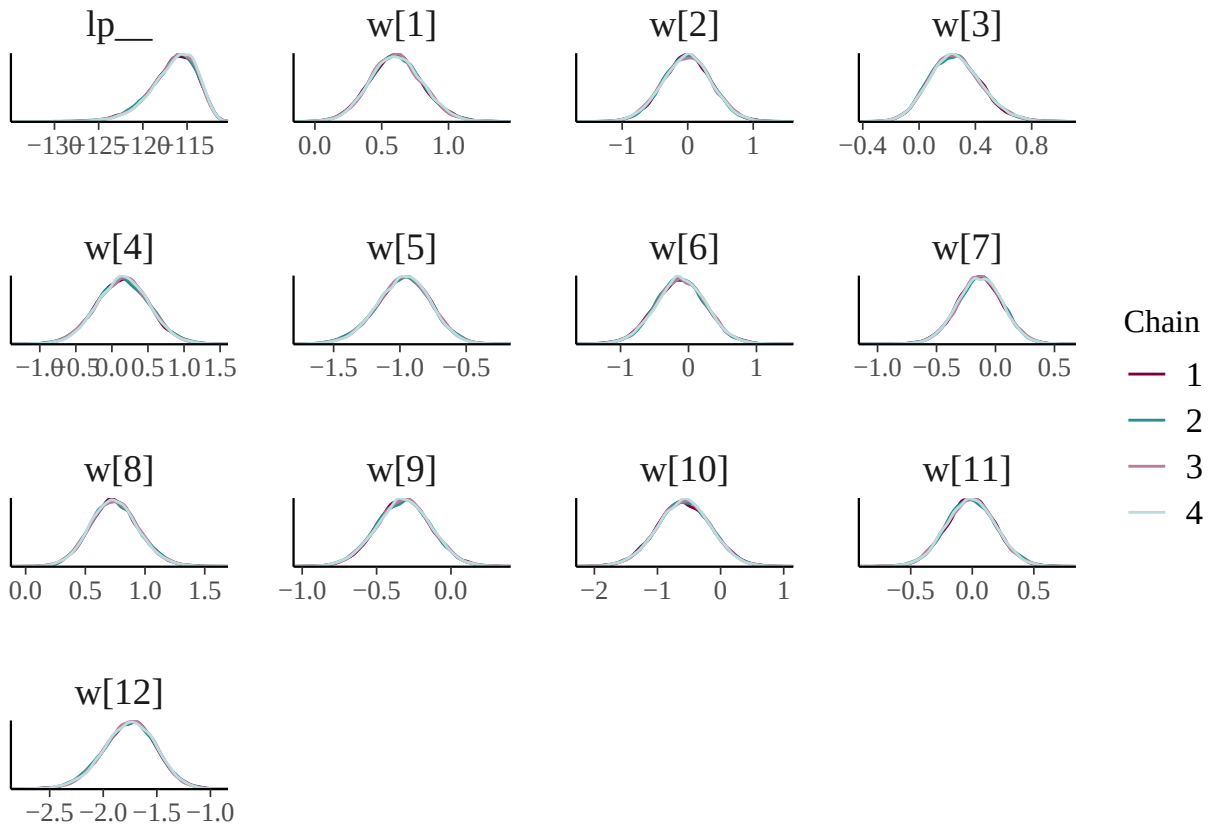
```
# A tibble: 5 x 5
  variable      mean  rhat ess_bulk ess_tail
  <chr>        <dbl> <dbl>    <dbl>    <dbl>
1 lp__      -113.      1.00   15106.   24530.
2 w1[1]        0.661   1.00   41285.   30728.
3 w1[2]        0.0480  1.00   28289.   28625.
4 w1[3]        0.444   1.00   28809.   27835.
5 w1[4]        0.172   1.00   28947.   28189.
```

7) Convergence diagnostics

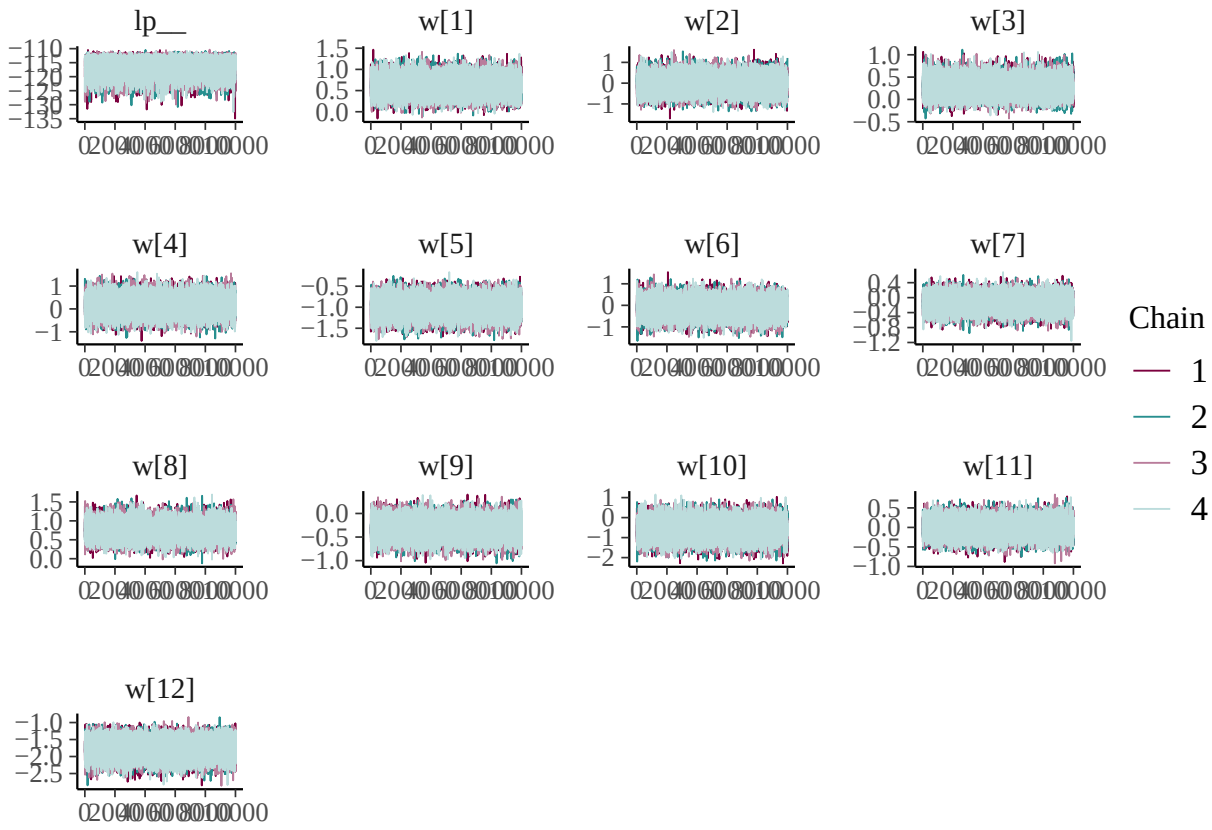
In order to assess the convergence of the models, we attempt to: - Check Rhat, ESS (effective sample size) values of the simulation to see whether the iterations chain mixed well together. These values could be checked from the model's summary table above. - Take a look at the density estimation of the parameters graphs to see whether they agrees with each other by overlapping - Look at the trace plot to see if the traces nearly overlaps

Looking at the summary table of both models, we can see the Rhat of value of all the chains are very near 1, and the ESS values are high. Furthermore, the weights estimation of the of the chains overlaps nicely, which can be see at the density and trace plots.

```
#Convergence diagnostics for the linear model
stan_linear_model_draws <- stan_linear_model_fit$draws()
parameters_of_concern <- c("lp__", "w[1]", "w[2]", "w[3]", "w[4]", "w[5]",
                           "w[6]", "w[7]", "w[8]", "w[9]", "w[10]",
                           "w[11]", "w[12]")
mcmc_dens_overlay(stan_linear_model_draws, pars = parameters_of_concern,
                  facet_args = list(ncol = 4)) +
                  facet_text(size = 14) #great to show convergence
```



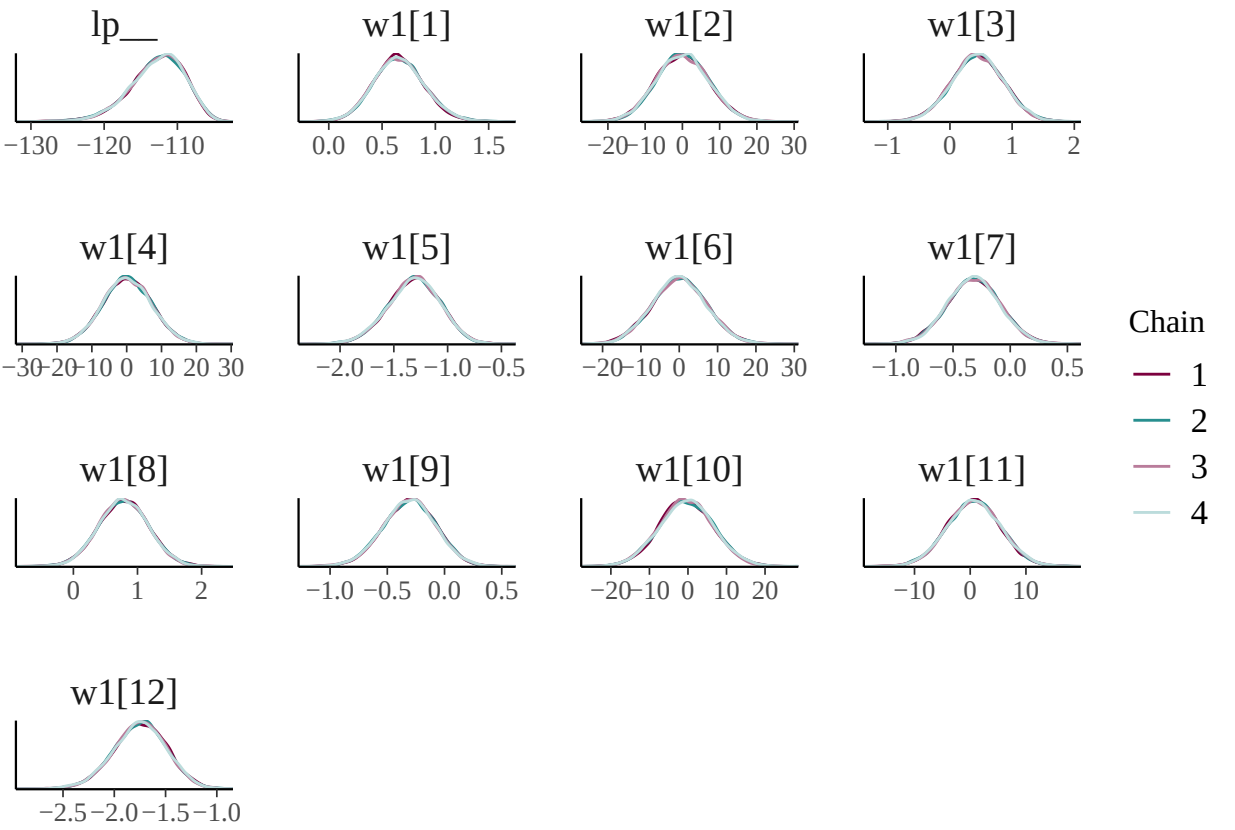

```
mcmc_trace(stan_linear_model_draws, pars = parameters_of_concern) #great to show convergence
```



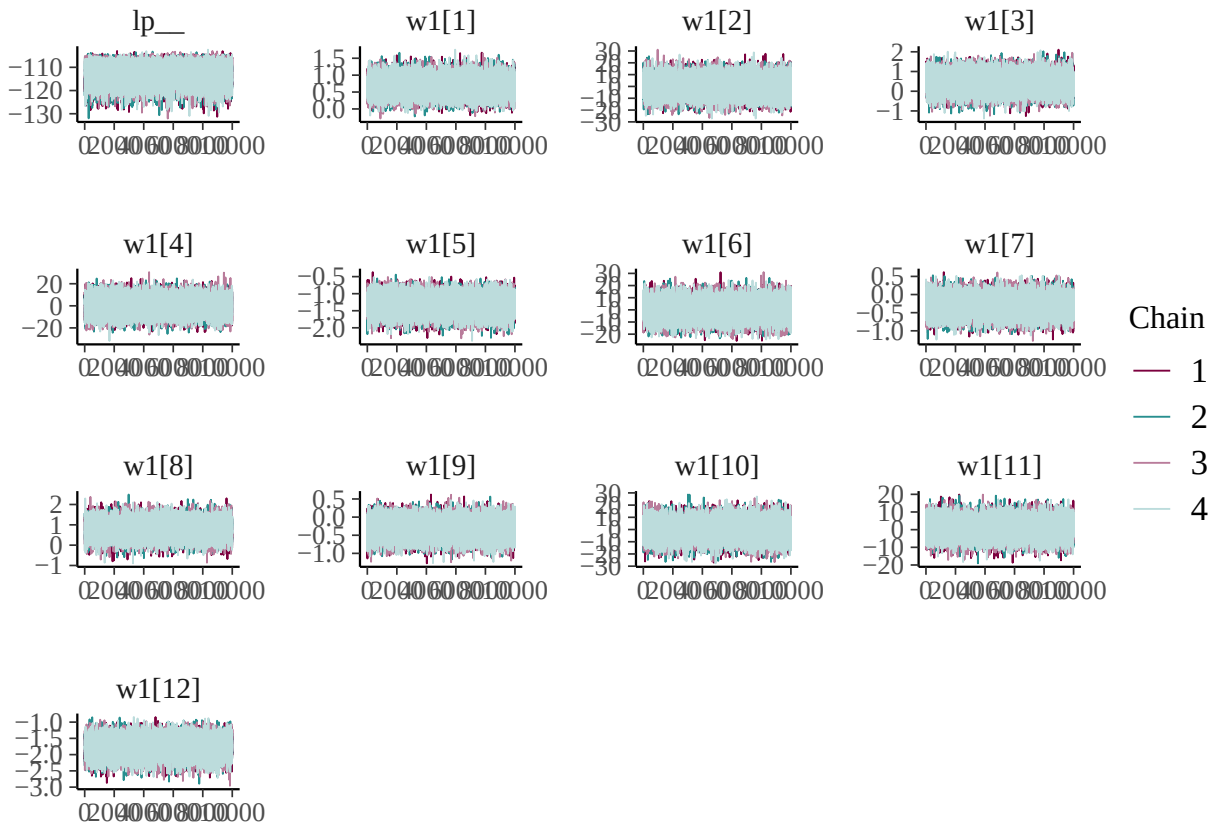

```
stan_linear_model_fit$diagnostic_summary()
```

```
$num_divergent
[1] 0 0 0 0

$num_max_treedepth
[1] 0 0 0 0

$ebfmi
[1] 0.9657418 1.0051887 0.9928076 1.0205822
```

```
#Convergence diagnostics for the 2nd order model
stan_model_2nd_order_draws <- stan_model_2nd_order_fit$draws()
parameters_of_concern <- c("lp__", "w1[1]", "w1[2]", "w1[3]", "w1[4]", "w1[5]",
                           "w1[6]", "w1[7]", "w1[8]", "w1[9]", "w1[10]",
                           "w1[11]", "w1[12]")
mcmc_dens_overlay(stan_model_2nd_order_draws, pars = parameters_of_concern,
                  facet_args = list(ncol = 4)) +
                  facet_text(size = 14) #great to show convergence
```




```
mcmc_trace(stan_model_2nd_order_draws, pars = parameters_of_concern) #great to show convergence
```




```
stan_model_2nd_order_fit$diagnostic_summary()
```

```
$num_divergent
[1] 0 0 0 0

$num_max_treedepth
[1] 0 0 0 0

$ebfmi
[1] 0.9698185 0.9286379 0.9551178 0.9620816
```

8) Posterior predictive check and classification accuracy

We made a stan_glm model with the same prior for predictive checks [6]

```
#Linear model formula
(reg_formula <- formula(paste("DEATH_EVENT ~",
                              paste(names(preprocessed_clinical_records)[1:(dim(preprocessed_clinical_r
                                    collapse = " + "))))
```

```
DEATH_EVENT ~ age + anaemia + creatinine_phosphokinase + diabetes +
    ejection_fraction + high_blood_pressure + platelets + serum_creatinine +
    serum_sodium + sex + smoking + time
```

```r
#Generalized linear model
post1 <- stan_glm(reg_formula, data = preprocessed_clinical_records,
                  family = binomial(link = "logit"), prior = normal(0,10),
                  prior_intercept = normal(0,10), QR=TRUE)
```

```r
#Leave-one-out
(loo1 <- loo(post1, save_psis = TRUE))
```

```
Computed from 4000 by 299 log-likelihood matrix

         Estimate   SE
elpd_loo   -124.7 11.1
p_loo        15.0  1.8
looic       249.4 22.2
------
Monte Carlo SE of elpd_loo is 0.1.

Pareto k diagnostic values:
                         Count Pct.    Min. n_eff
(-Inf, 0.5]   (good)     297   99.3%   1055
 (0.5, 0.7]   (ok)         2    0.7%   3077
   (0.7, 1]   (bad)        0    0.0%   <NA>
   (1, Inf)   (very bad)   0    0.0%   <NA>

All Pareto k estimates are ok (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

```r
#Predictive performance measures
preprocessed_clinical_records_2 <- preprocessed_clinical_records
preprocessed_clinical_records_2$DEATH_EVENT <- factor(preprocessed_clinical_records_2$DEATH_EVENT)
linpred <- posterior_linpred(post1)
preds <- posterior_epred(post1)
pred <- colMeans(preds)
pr <- as.integer(pred >= 0.5)
y <- preprocessed_clinical_records_2$DEATH_EVENT
x <- model.matrix(DEATH_EVENT ~ . - 1, data = preprocessed_clinical_records)
```

```r
# posterior classification accuracy
round(mean(xor(pr,as.integer(y==0))),2)
```

```
[1] 0.86
```

```r
# posterior balanced classification accuracy
round((mean(xor(pr[y==0]>0.5,as.integer(y[y==0])))+mean(xor(pr[y==1]<0.5,as.integer(y[y==1]))))/2,2)
```

```
[1] 0.82
```

```r
# LOO predictive probabilities
ploo=E_loo(preds, loo1$psis_object, type="mean", log_ratios = -log_lik(post1))$value
# LOO classification accuracy
round(mean(xor(ploo>0.5,as.integer(y==0))),2)
```

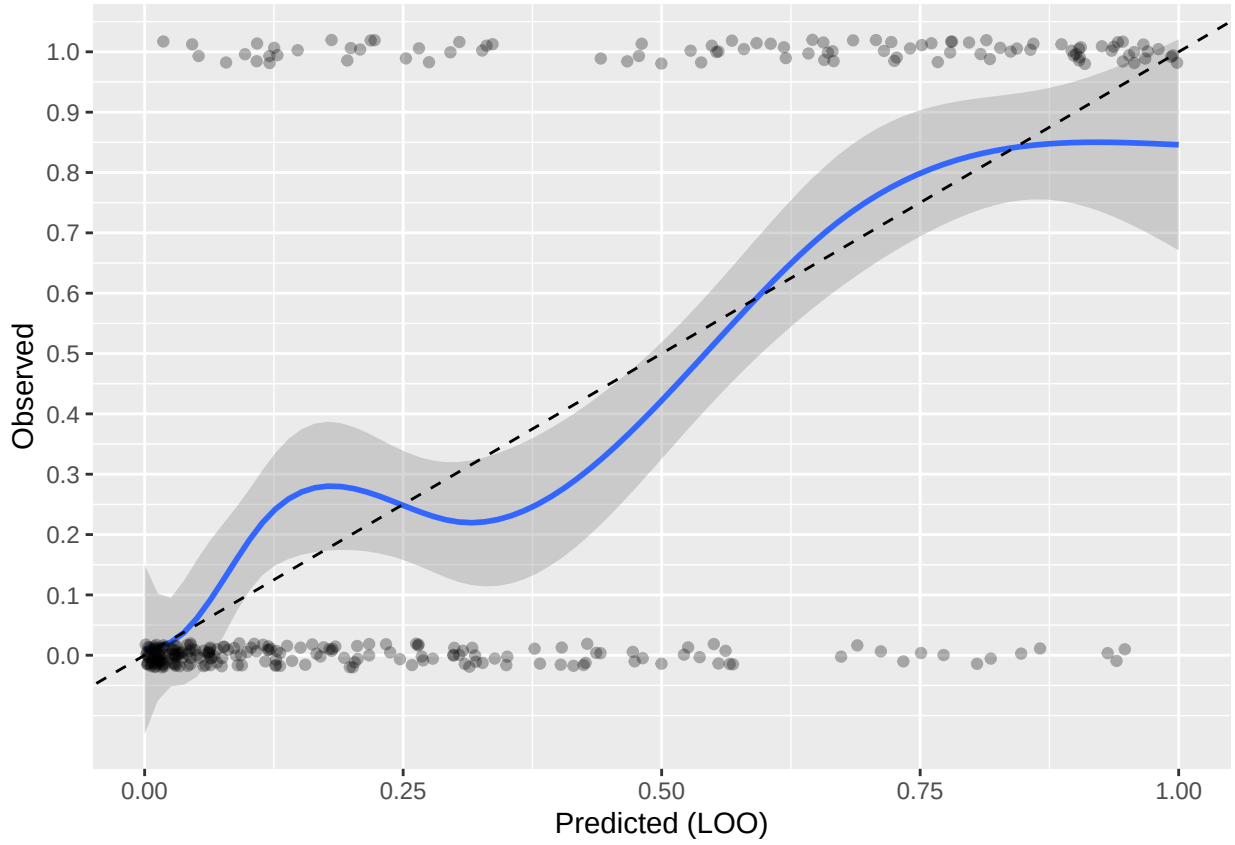
```
[1] 0.83
```

```
# LOO balanced classification accuracy
round((mean(xor(ploo[y==0]>0.5,as.integer(y[y==0])))+mean(xor(ploo[y==1]<0.5,as.integer(y[y==1]))))/2,2)
```

```
[1] 0.79
```

Classification accuracies seem to be decent.

```
ggplot(data = data.frame(pred=pred,loopred=ploo,y=as.numeric(y)-1), aes(x=loopred, y=y)) +
  stat_smooth(method='glm', formula = y ~ ns(x, 5), fullrange=TRUE) +
  geom_abline(linetype = 'dashed') +
  labs(x = "Predicted (LOO)", y = "Observed") +
  geom_jitter(height=0.02, width=0, alpha=0.3) +
  scale_y_continuous(breaks=seq(0,1,by=0.1)) +
  xlim(c(0,1))
```



From the plot above, we can see that model doesn't perfectly predict the value. There are parts where the model predicts less instances than observed ([0.00,0.25] and [0.60,0.80] on the x-axis) and also predicts more instances than observed ([0.25,0.60] and [0.80,1.00] on the x-axis). In total it predicts more instances than observed (0.8 observed vs 1.00 predicted).

9) Sensitivity analysis

The prior sensitivity can be checked by using a relatively different prior to fit the model and see whether the final results differs. If the final result stays the same then the model is not sensitive to prior (unless the new prior is very bad). In our situation, we feed different means and std as priors from the original model and draw the density graph of weights in fitted model and compares them. As the final fitted weights w of their respective model are nearly identical, we can says that these models are prior-insensitive. Below we showcase a few weights fitted from the models, with the top rows are model that used the original priors, and the bottom rows are the model that used alternate priors. As you can see, their respective densities curves are nearly identical.

```
#Fit linear model with alternate prior (for prior sensitivity analysis)
alternate_weights_prior_mean <- c(4,8,-8.1,-16,5,3.5,6,-8,1,7,9,6)
alternate_weights_prior_std <- c(5,8,8,7.8,7,9.1,6,8,12,10,7,8)
alternate_bias_prior_mean <- 3
alternate_bias_prior_std <- 5

alternate_prior_stan_linear_model_fit <- fit_linear_model(weights_prior_mean = alternate_weights_prior_m
                                    weights_prior_std = alternate_weights_prior_std,
                                    bias_prior_mean = alternate_bias_prior_mean,
                                    bias_prior_std = alternate_bias_prior_std)
```

```
Running MCMC with 4 parallel chains...

Chain 4 finished in 19.3 seconds.
Chain 2 finished in 19.4 seconds.
Chain 1 finished in 19.8 seconds.
Chain 3 finished in 19.8 seconds.

All 4 chains finished successfully.
Mean chain execution time: 19.6 seconds.
Total execution time: 19.8 seconds.
```

```
alternate_prior_stan_model_draws <- alternate_prior_stan_linear_model_fit$draws()
```

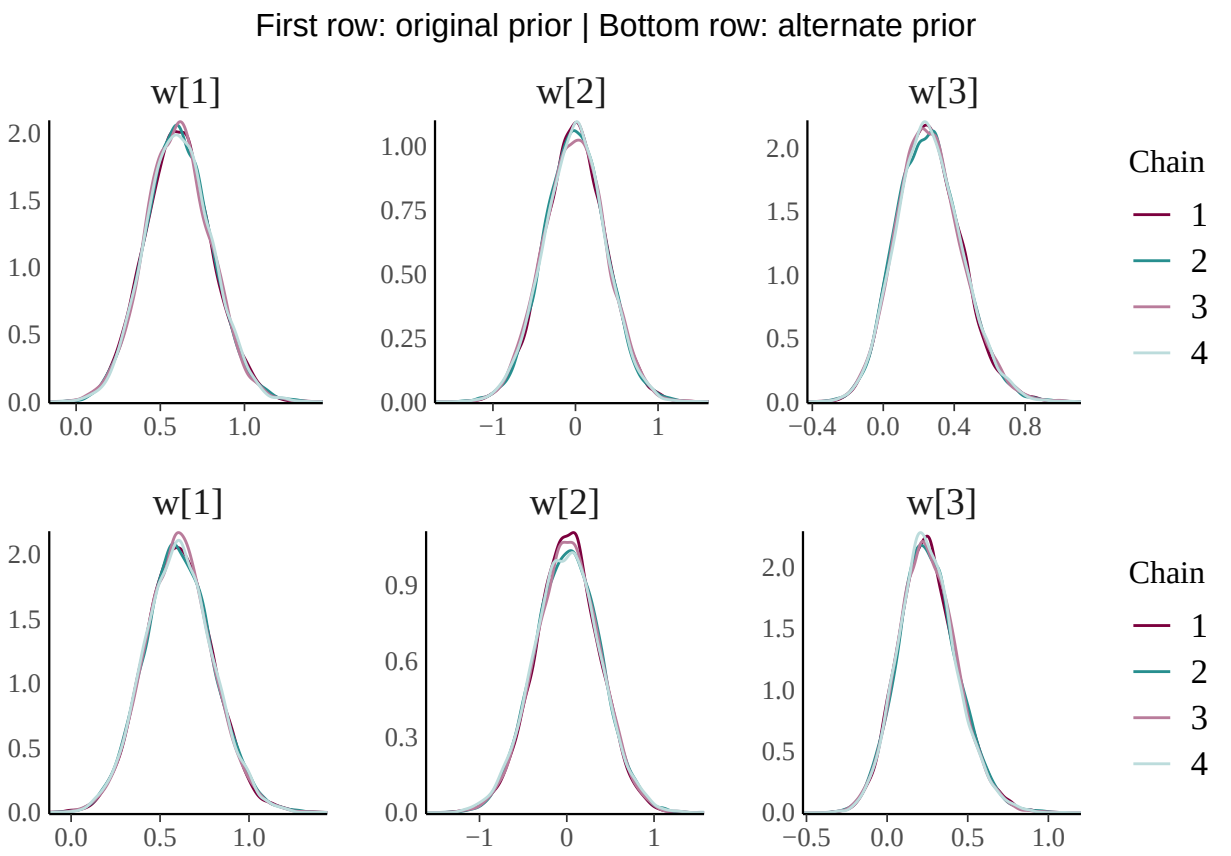
```
#Fit 2nd order model with alternate prior (for prior sensitivity analysis)
weights_1st_order_alternate_prior_mean <- c(3,5,3,5,3,3,2,2,3,4,5,6)
weights_1st_alternate_prior_std <- c(8,9,7,10,10,10,10,10,10,10,10,10)
weights_2nd_order_alternate_prior_mean <- c(-2,5,-4,-8,1,3,5,2,-6,8,5,-9)
weights_2nd_alternate_prior_std <- c(10,10,10,10,10,10,10,10,10,10,10,10)
bias_alternate_prior_mean <- 5
bias_alternate_prior_std <- 10

stan_model_2nd_order_alt_prior_fit <- 0
if(file.exists("stan_model_2nd_order_alt_prior_fit.RDS")) {
  stan_model_2nd_order_alt_prior_fit <- readRDS(file = "stan_model_2nd_order_alt_prior_fit.RDS")
} else {
  stan_model_2nd_order_alt_prior_fit <- fit_2nd_order_model(
                weights_1st_order_prior_mean = weights_1st_order_alternate_prior_mean,
                weights_1st_prior_std = weights_1st_alternate_prior_std,
                weights_2nd_order_prior_mean = weights_2nd_order_alternate_prior_mean,
                weights_2nd_prior_std = weights_2nd_alternate_prior_std,
```

```
                bias_prior_mean = bias_prior_mean,
                bias_prior_std = bias_prior_std)
  stan_model_2nd_order_alt_prior_fit$save_object(file = "stan_model_2nd_order_alt_prior_fit.RDS")
}

stan_model_2nd_order_alt_prior_draws <- stan_model_2nd_order_alt_prior_fit$draws()
```

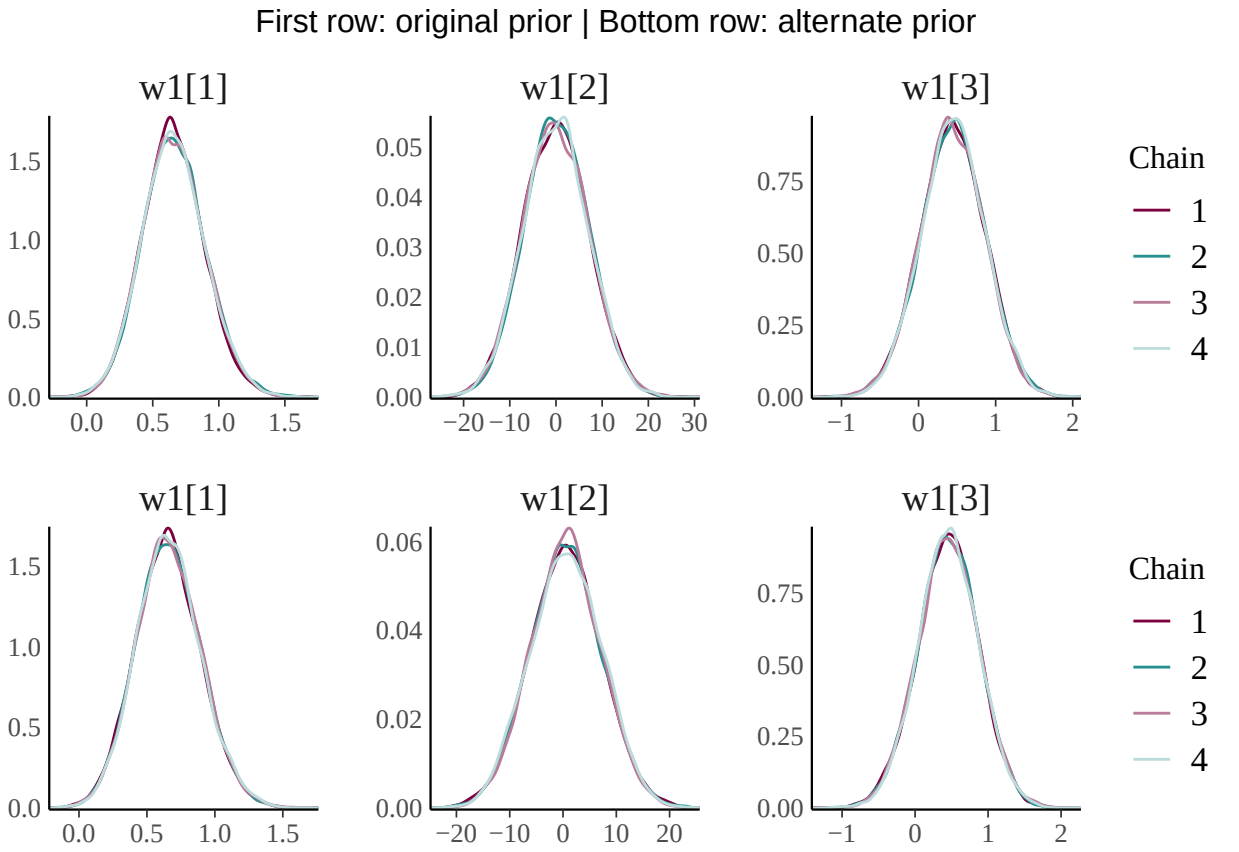
```
#Function to draws the fitted parameters for visual comparision
compare_models_result_visual <- function(model_og_draws,
                                         model_alt_draws,
                                         params){
  w_a <- mcmc_dens_overlay(model_og_draws, pars = params,
                facet_args = list(ncol = 3)) + facet_text(size = 14) + yaxis_text()
  w_b <- mcmc_dens_overlay(model_alt_draws, pars = params,
                facet_args = list(ncol = 3)) + facet_text(size = 14) + yaxis_text()
  grid.arrange(w_a, w_b, ncol=1,
               top="First row: original prior | Bottom row: alternate prior")
}
```

Compare the fitted parameters visually. We can see that ...

```
compare_models_result_visual(stan_linear_model_draws,
                             alternate_prior_stan_model_draws,
                             c("w[1]", "w[2]", "w[3]"))
```



```
#Visually check whether the 2nd order model converges
compare_models_result_visual(stan_model_2nd_order_draws,
                             stan_model_2nd_order_alt_prior_draws,
                             c("w1[1]", "w1[2]", "w1[3]"))
```



10) Model comparision

We compare the model in these two aspects: - Whether the model reliable with k-pareto. - Compare the loo elpd (leave-one-out cross validation expected log predictive density). The model has higher value is better.

In the code below we see that all the k-pareto value of the linear model is good, but there are a some k-value of the quadratic model goes above the good threshold. This means that the quadratic model should not be considered reliable, and in this case we guess it likely to exhibit over fitting (similar to when we try to fit model as good as possible with high degree polynomial). Comparing the elpd indicates that the better candidate is the linear model.

```
stan_linear_model_fit_loo <- stan_linear_model_fit$loo()
stan_linear_model_fit_loo
```

```
Computed from 40000 by 299 log-likelihood matrix

         Estimate   SE
elpd_loo   -125.1 11.1
```

```
p_loo        15.4  1.9
looic       250.1 22.2
------
Monte Carlo SE of elpd_loo is 0.0.

Pareto k diagnostic values:
                         Count Pct.    Min. n_eff
(-Inf, 0.5]   (good)     297   99.3%   12706
 (0.5, 0.7]   (ok)         2    0.7%   6000
   (0.7, 1]   (bad)        0    0.0%   <NA>
   (1, Inf)   (very bad)   0    0.0%   <NA>

All Pareto k estimates are ok (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

```
stan_model_2nd_order_fit_loo <- stan_model_2nd_order_fit$loo()
stan_model_2nd_order_fit_loo
```

```
Computed from 40000 by 299 log-likelihood matrix

         Estimate   SE
elpd_loo   -125.6 12.6
p_loo        25.4  3.4
looic       251.2 25.3
------
Monte Carlo SE of elpd_loo is NA.

Pareto k diagnostic values:
                         Count Pct.    Min. n_eff
(-Inf, 0.5]   (good)     292   97.7%   3915
 (0.5, 0.7]   (ok)         4    1.3%   488
   (0.7, 1]   (bad)        3    1.0%   193
   (1, Inf)   (very bad)   0    0.0%   <NA>
See help('pareto-k-diagnostic') for details.
```

```
#Compare the two models with loo (expected log pointwise predictive density with leave-one-out cv)
loo::loo_compare(stan_linear_model_fit_loo, stan_model_2nd_order_fit_loo)
```

```
       elpd_diff se_diff
model1  0.0       0.0
model2 -0.5       5.3
```

```
#Compare our stan_glm model with our stan_linear_model
loo::loo_compare(stan_linear_model_fit_loo, loo1)
```

```
       elpd_diff se_diff
post1   0.0       0.0
model1 -0.3       0.1
```

We can see there's very minimal difference between stan_linear and stan_glm.

11) Discussion of issues and potential improvements

Our group could have done a lot more and we could have gone considerably more in-depth. For instance, the two models that were used in the project were a linear model and a non-linear model. Perhaps, a hierarchical model could have been employed. There was also a big correlation between two clinical features, smoking and sex. This could be seen from the correlation plot of features in the first chapters. Many pieces of research suggest that men are more likely to smoke than women but the dependency needs to be resolved for better analysis.

Another option could have been to experiment with certain types of machine learning models. We could have tested the accuracy of a machine learning model and compared it with the linear and non-linear model in our report. Possibly a larger than two polynomials could have performed better than the two polynomial.

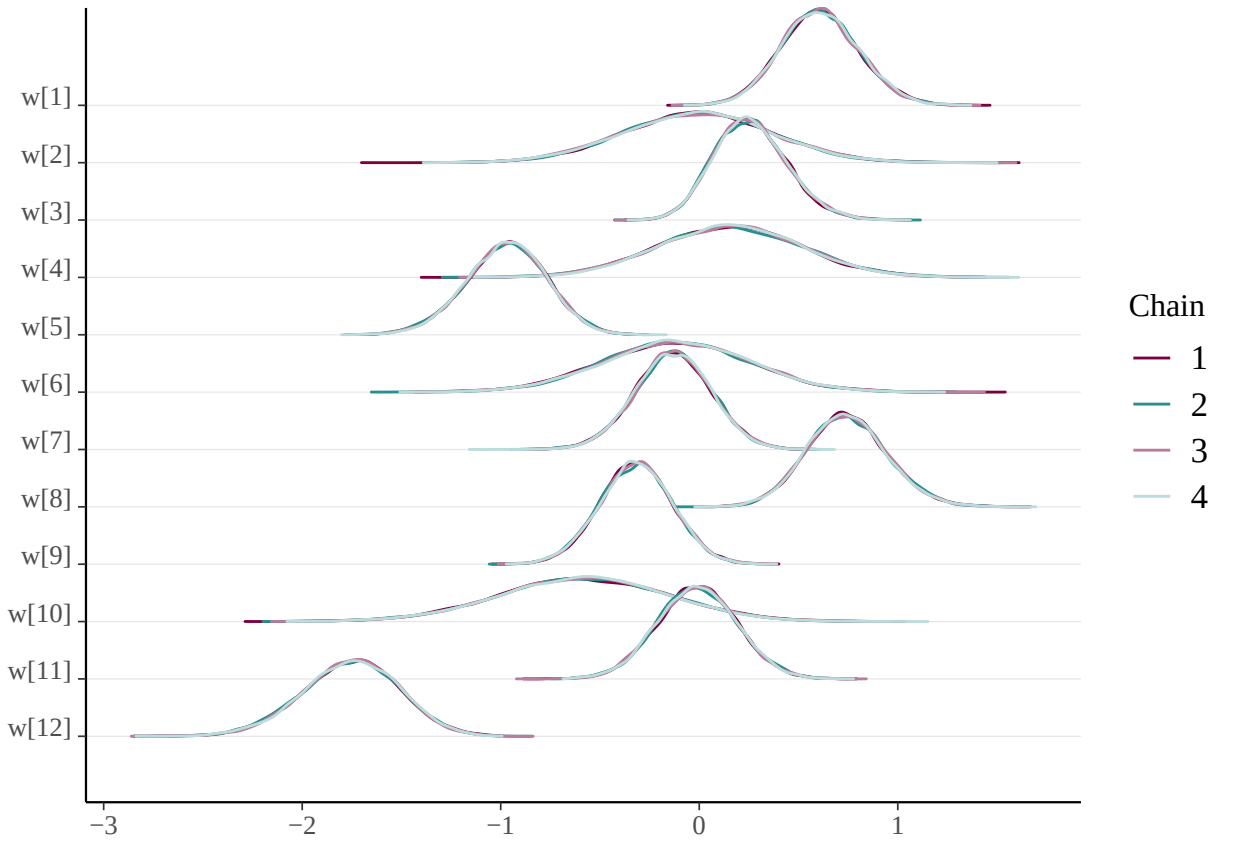
We got our feature scaling working with the linear model, but it's not too good for the quadratic model. For the linear model we got reasonable fitting times within 1 minute, but for the quadratic it took 10 minutes to fit, which is too long and should be improved.

12) Conclusion what was learned from the data analysis

Below is a brief conclusion about what was learnt from the data analysis. The conclusion only goes through the important aspects and the rest of the information can be found in the individual chapters.

- Rhat of value each chain in convergence diagnostic essentially 1 and Effective sample size values are large indicating that our model converges and we can be satisfied with the simulations.
- Posterior predictive check tells us that model doesn't perfectly predict the value. Classification accuracy appears to be respectable at around 0.86.
- Same models with different priors are nearly identical. Indicating our models are prior-insensitive.
- For predicting the death event, our model shows that weights for time (follow up period) (w[12]) and ejection fraction (w[5]) have high negative correlation with the death event, and w1, w8 have a high positive correlation with the death event according to the density graph in chapter 10. Both high positive and negative values diverge from 0. With normal logic we can see that the older the person is (positive correlation with death), and the less follow days (negative correlations with death, the person dies soon after getting cared), the likelihood the patient dies increase. However, the serum creatinine and ejection fraction are the value we can measured and make use for prediction. Looking at these correlation, We can also have a rough common sense explanation for low ejection fraction, the person likely to die since the heart is cannot pump enough blood. These pairs of death event correlation, "serum creatinine" and "ejection fraction" actually coincidence with the article where the dataset originates from: "Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone".

```
parameters_of_concern <- c("w[1]", "w[2]", "w[3]", "w[4]", "w[5]",
                           "w[6]", "w[7]", "w[8]", "w[9]", "w[10]",
                           "w[11]", "w[12]")
mcmc_dens_chains(stan_linear_model_draws, pars = parameters_of_concern)
```



13) Self-reflection of what the group learned while making the project

Our group learnt a lot of new things in the context of data analysis and information visualisation. Aspects like the correct way of referencing will make a prominent difference in the final report. Furthermore, things like working together in a group and having to schedule to the advantage of the group reassemble reality. All in all, there were plenty of new specialities and techniques learnt that will most probably be beneficial for us when doing similar type of assignments or jobs in the future.

References

[DATASET] Davide Chicco, Giuseppe Jurman (2020): "Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. BMC Medical Informatics and Decision Making" 20, 16. (https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-1023-5)

[1] "Cardiovascular diseases (CVDs)." https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds). [accessed 2.12.2022]

[2] "Creatine phosphokinase test." https://www.mountsinai.org/health-library/tests/creatine-phosphokinase-test. [accessed 2.12.2022] [3] "Ejection Fraction." https://my.clevelandclinic.org/health/articles/16950-ejection-fraction. [accessed 2.12.2022]

[4] "Serum creatinine test." https://www.kidneyfund.org/all-about-kidneys/tests/serum-creatinine-test. [accessed 2.12.2022]

[5] "serum sodium." https://medlineplus.gov/lab-tests/sodium-blood-test/serum_sodium/ [accessed 2.12.2022]

[6] Bayesian logistic regression model: https://avehtari.github.io/modelselection/diabetes.html#4_A_Bayesian_logistic_regression_model

[7] https://www.kaggle.com/code/monogenea/bayesian-approach-to-heart-failure-prediction