Multiple linear regression prediction

Let Y_i be the percent increase in GOP support from 2012 to 2016 in county $i = 1, \ldots, n$. We model

$$Y_i|\beta, \sigma^2 \sim \text{Normal}(\alpha + X_{1i}\beta_1 + \ldots + X_{pi}\beta_p, \sigma^2)$$

where X_{ji} s the j^{th} covariate for county i. All variables are centered and scaled. We select prior $\sigma^2 \sim \text{InvGamma}(0.01, 0.01)$ and $\alpha \sim \text{Normal}(0, 100)$ for the error variance and intercept, and compare different priors for the regression coefficients.

Load and standardize the election data

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

```
 set.seed(1111)
# Identify rows with NA in either Y or any column of X
complete_rows <- complete.cases(Y, X)
Y <- Y[complete_rows]
X <- X[complete_rows,]
 n    = length(Y)
 p    = ncol(X)
 n
```

```
## [1] 3111
```

```
 p
```

```
## [1] 10
```

```
#Scaling input features
 X    = scale(X)
#Fit the model by using sample size of 100 datasets for the training and use the remaining as a test da
# Generate a random permutation of indices from 1 to n
indices =sample(n)
# Logical vector indicating whether the index is greater than 100
test=indices > 100
# Create a table of the logical vector
table(test)
```

```
## test
## FALSE  TRUE
##   100  3011
```

```r
# Train data
Y_train    = Y[!test]
X_train    = X[!test,]
#Test data
Y_test    = Y[test]
X_test    = X[test,]
n_train    = length(Y_train)
n_test    = length(Y_test)
p     = ncol(X_train)
```

Fit the linear regression model with Gaussian priors

```r
model_string = "model{

  # Likelihood
  for(i in 1:n_train){
    Y_train[i]   ~ dnorm(muo[i],inv.var)
    muo[i] <- alpha + inprod(X_train[i,],beta[])
  }

  # Prediction
  for(i in 1:n_test){
    Y_test[i]  ~ dnorm(mup[i],inv.var)
    mup[i] <- alpha + inprod(X_test[i,],beta[])
  }

  # Priors
  for(j in 1:p){
    beta[j] ~ dnorm(0,0.0001)
  }
  alpha     ~ dnorm(0, 0.01)
  inv.var   ~ dgamma(0.01, 0.01)
  sigma     <- 1/sqrt(inv.var)
}"
```

Compile the model in JAGS

```r
model = jags.model(textConnection(model_string),
                    data = list(Y_train=Y_train,n_train=n_train,n_test=n_test,p=p,X_train=X_train,X_tes
```

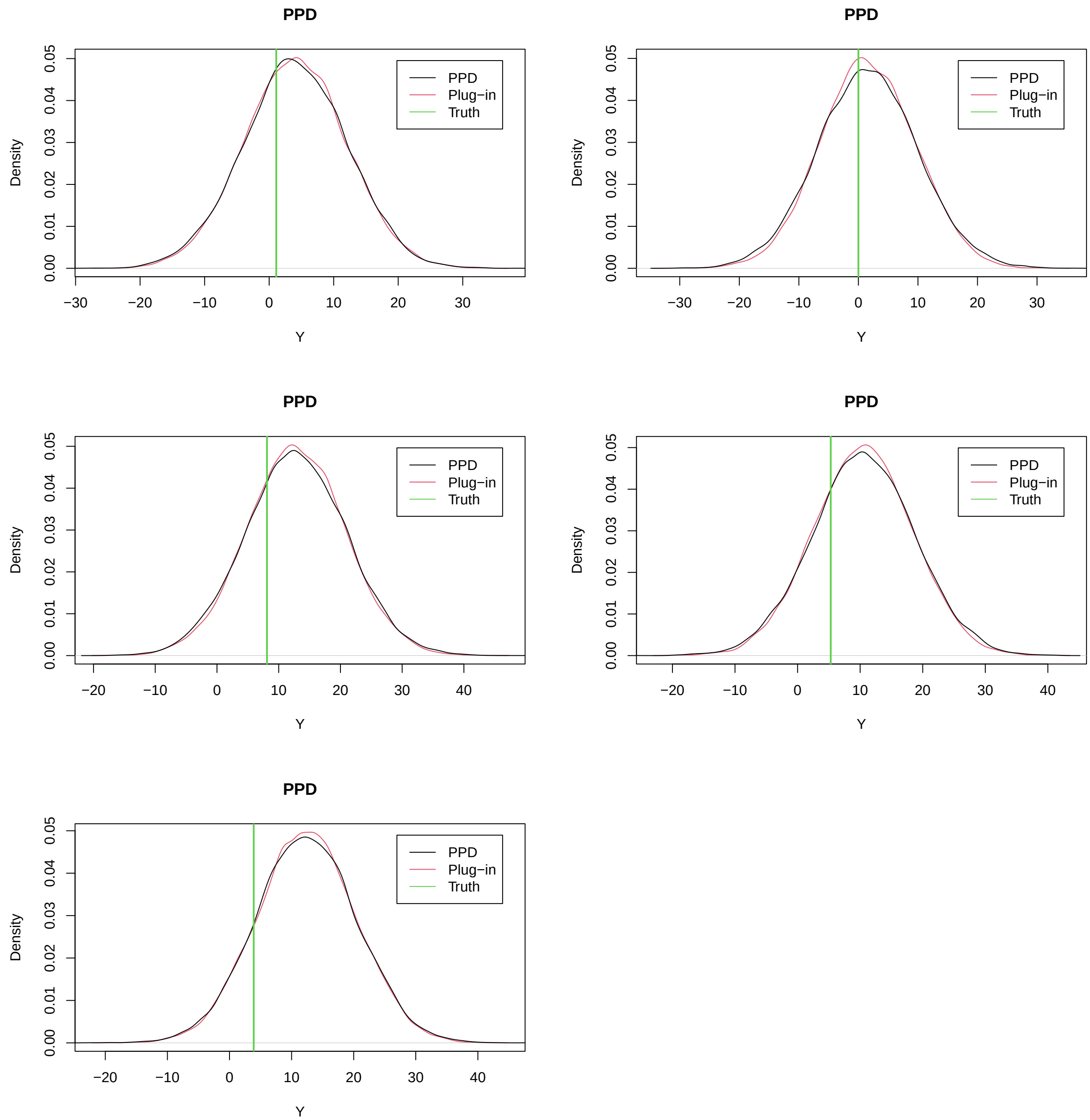
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 3023
##    Total graph size: 43576
##
## Initializing model
```

```r
update(model, 10000, progress.bar="none")

samp = coda.samples(model,
        variable.names=c("beta","sigma","Y_test","alpha"),
        n.iter=20000, progress.bar="none")

summary(samp[,-c(1:n_test)])
```

```
##
## Iterations = 10001:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean     SD Naive SE Time-series SE
## alpha     7.7408 0.8658 0.006122       0.007875
## beta[1]  -1.1715 1.0282 0.007271       0.011513
## beta[2]   5.5726 1.5495 0.010957       0.026342
## beta[3]   0.1723 0.9494 0.006713       0.011599
## beta[4]  -2.2915 1.0764 0.007612       0.015466
## beta[5]  -1.1315 1.4475 0.010236       0.025024
## beta[6]  -5.6806 1.6731 0.011830       0.030790
## beta[7]  -0.2829 1.2159 0.008598       0.017133
## beta[8]  -0.9986 1.7063 0.012065       0.036323
## beta[9]   2.6540 2.1820 0.015429       0.057316
## beta[10]  0.5607 1.9110 0.013513       0.041035
## sigma     7.8995 0.5975 0.004225       0.005054
##
## 2. Quantiles for each variable:
##
##            2.5%     25%     50%     75%   97.5%
## alpha     6.011  7.1692  7.7400  8.3294  9.4208
## beta[1]  -3.214 -1.8549 -1.1709 -0.4787  0.8255
## beta[2]   2.592  4.5058  5.5646  6.6086  8.6477
## beta[3]  -1.668 -0.4754  0.1699  0.8122  2.0461
## beta[4]  -4.397 -3.0192 -2.2834 -1.5823 -0.1741
## beta[5]  -3.948 -2.0928 -1.1433 -0.1702  1.7157
## beta[6]  -8.928 -6.8046 -5.6873 -4.5664 -2.3955
## beta[7]  -2.667 -1.1003 -0.2882  0.5388  2.1114
## beta[8]  -4.367 -2.1191 -0.9913  0.1334  2.3659
## beta[9]  -1.641  1.1870  2.6583  4.1146  6.9132
## beta[10] -3.192 -0.7060  0.5372  1.8526  4.3252
## sigma     6.836  7.4796  7.8621  8.2776  9.1897
```

Plot samples from the posterior preditive distribution (PPD) and plug-in distribution

```
# Extract the samples for each parameter
# Extract the samples for each parameter
samps = as.matrix(samp)
Y_test_samps = samps[, grep("Y_test", colnames(samps))]
alpha_samps = samps[, "alpha"]
beta_samps = samps[, grep("beta", colnames(samps))]
sigma_samps = samps[, "sigma"]

# Compute the posterior mean for the plug-in predictions
compute_posterior_means <- function(alpha_samps, beta_samps, sigma_samps) {
  list(
    alpha_mn = mean(alpha_samps),
    beta_mn = colMeans(beta_samps),
    sigma_mn = mean(sigma_samps)
  )
}

posterior_means <- compute_posterior_means(alpha_samps, beta_samps, sigma_samps)


# Plot the PPD and plug-in
plot_ppd_and_plugin <- function(X_test, Y_test, Y_test_samps, posterior_means, index) {
  alpha_mn = posterior_means$alpha_mn
  beta_mn = posterior_means$beta_mn
  sigma_mn = posterior_means$sigma_mn

  mu = alpha_mn + sum(X_test[index, ] * beta_mn)
  y = rnorm(20000, mu, sigma_mn)

  plot(density(y), col = 2, xlab = "Y", main = "PPD")
  lines(density(Y_test_samps[, index]))
  abline(v = Y_test[index], col = 3, lwd = 2)
  legend("topright", c("PPD", "Plug-in", "Truth"), col = 1:3, lty = 1, inset = 0.05)
}

for (j in 1:5) {
  plot_ppd_and_plugin(X_test, Y_test, Y_test_samps, posterior_means, j)
}
```




```r
# 95% intervals with plug-in approach
alpha_mn <- posterior_means$alpha_mn
beta_mn <- posterior_means$beta_mn
sigma_mn <- posterior_means$sigma_mn

low1 = alpha_mn + X_test %*% beta_mn - 1.96 * sigma_mn
high1 = alpha_mn + X_test %*% beta_mn + 1.96 * sigma_mn
cover1 = mean(Y_test > low1 & Y_test < high1)
mean(cover1)
```

```
## [1] 0.9262703
```

```r
# 95% intervals with PPD
low2 = apply(Y_test_samps, 2, quantile, 0.025)
high2 = apply(Y_test_samps, 2, quantile, 0.975)
cover2 = mean(Y_test > low2 & Y_test < high2)
mean(cover2)
```

```
## [1] 0.9468615
```

PPD densities are moderately wider than the plug-in densities. It is expected. Since, this is the effect of accounting for uncertainty in β and σ, and it explains the slightly lower covarage for the plug-in predictions.